# Optimal trees and forests

**Filip Malmberg**

# Todays lecture

- Trees and forests
- Optimal forests
    - Minimum spanning forests
    - Shortest path forests
- Applications in image segmentation

# Part 1: Forests and trees

# Forests and trees

In this lecture, we will consider two special types of graphs: *forests* and *trees*.

- A forest is a graph without simple cycles.
- A tree is a connected forest

(In other words, a forest is a collection of trees)

# Recall: Cycles, connected graphs

- A *cycle* is a path where the start vertex is the same as the end vertex.
- A cycle is *simple* if it has no repeated vertices other than the endpoints.
- Two vertices $v, w \in V$ are *linked* if $G$ contains a path from $v$ to $w$.
- A graph is *connected* if every pair of vertices on the graph is linked.
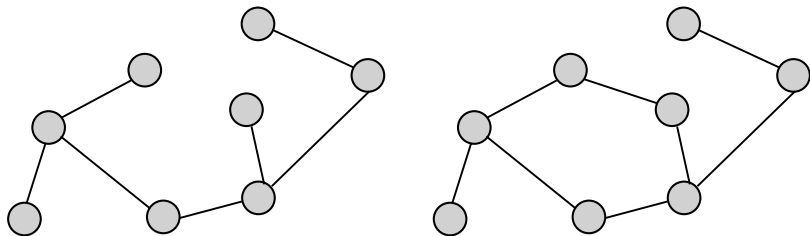
# Tree, example



Figure 1: Left: A tree. Right: Not a tree.

# Properties of trees and forests

- There is a *unique* path between each (linked) pair of vertices. *Why?*
- Any subset of the edges of a forest is a cut. *Why?*

# Recall: Cuts

- Let $S \subseteq E$, and $G' = (V, E \setminus S)$. If, for all $e_{v,w} \in S$, it holds that $v \underset{G'}{\not\sim} w$, then $S$ is a *(graph) cut* on $G$.

# Spanning trees

## Definition, spanning tree

Let $G$ be a connected, undirected graph. Let $T$ be a subgraph of $G$ such that

- $T$ is a tree.
- $V(T) = V(G)$.

Then $T$ is a *spanning tree* of $G$.

For any $G$, there exists at least one spanning tree. *Why?*

# Edge weighted graphs

- We associate each edge $e \in E$ with a real valued, non-negative *weight*, $w(e)$.
- The weight of an edge represents the dissimilarity (or, alternatively, similarity) between the vertices connected by the edge.
- For example, we may define the edge weights as

$$w(e_{ij}) = |I(v) - I(j)| , \tag{1}$$

where $I(v)$ is the intensity of the image element corresponding to $v$.

# Part 2: Minimum spanning trees

# Minimum spanning trees

- A graph can have many different spanning trees. A *minimum spanning tree* (MST) is a spanning tree $T = (V, E')$ that (globally) minimizes

$$f(T) = \sum_{e \in E'} w(e) . \tag{2}$$

- Although this is a global optimization problem, efficient algorithms for computing minimum spanning trees exist. We will now take a look at two such algorithms: Prim's algorithm [7] and Kruskal's algorithm [6].

# Kruskal's algorithm

## Kruskal's algorithm

Set $E_{new} = \emptyset$.
**while** *there exists an edge $e_{p,q}$ such that $p \not\sim_{(V, E_{new})} g$* **do**
| Choose such an edge with minimal weight and add it to $E_{new}$.
**end**

- At the termination of the algorithm, $(V, E_{new})$ is a MST on $G$.

# Kruskal's algorithm, example
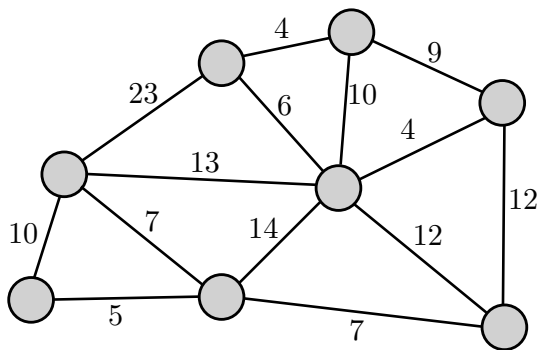


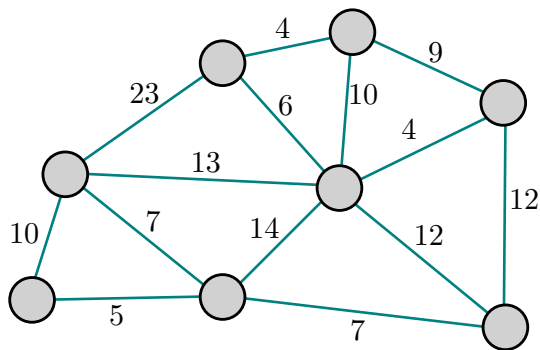Figure 2: An edge weighted graph.

# Kruskal's algorithm, example



Figure 3: Choose an edge with minimal weight that does not form a cycle.

# Kruskal's algorithm, example
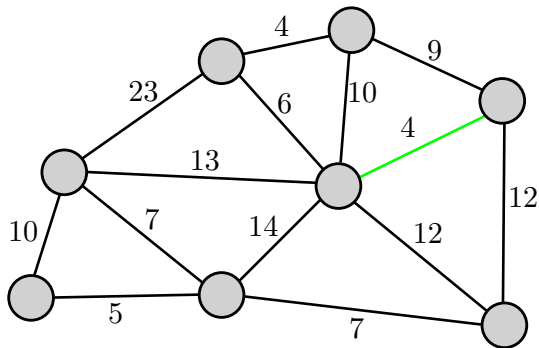


Figure 4: Add this edge to the tree.

# Kruskal's algorithm, example
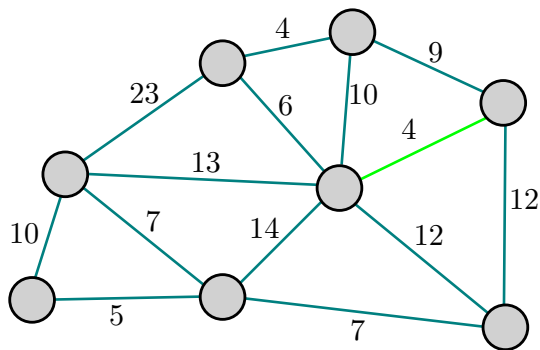


Figure 5: Choose an edge with minimal weight that does not form a cycle.
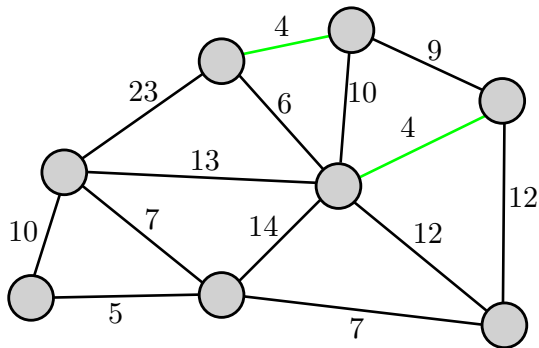
# Kruskal's algorithm, example



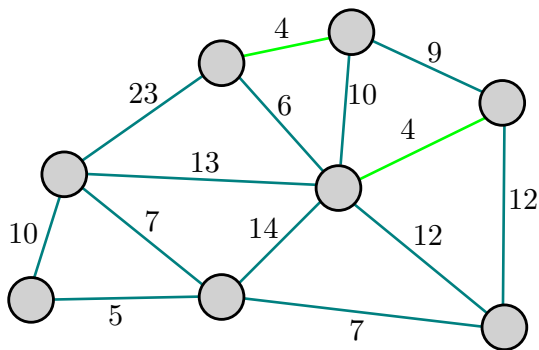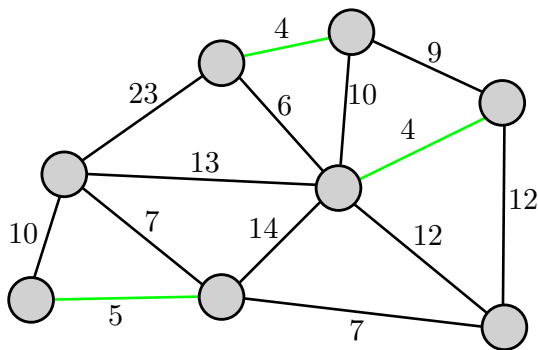Figure 6: Add this edge to the tree.

# Kruskal's algorithm, example
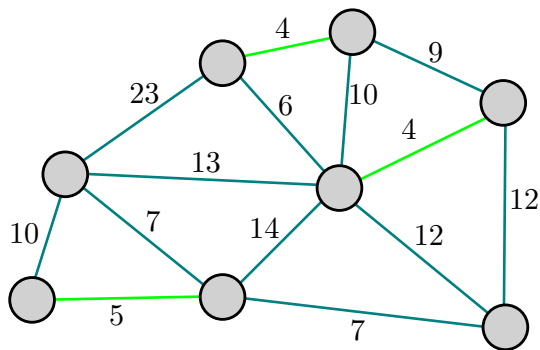


Figure 7: Choose an edge with minimal weight that does not form a cycle.

Centre for Image Analysis
Uppsala University

# Kruskal's algorithm, example



Figure 8: Add this edge to the tree.

# Kruskal's algorithm, example



Figure 9: Choose an edge with minimal weight that does not form a cycle.
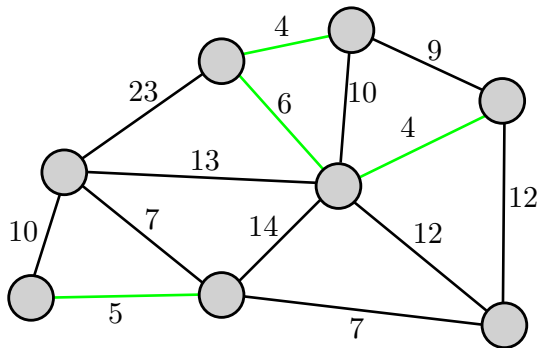
# Kruskal's algorithm, example



Figure 10: Add this edge to the tree.
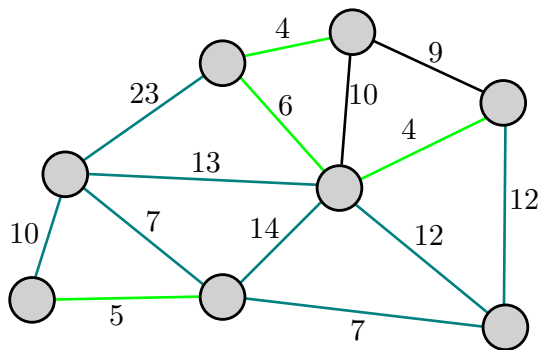
# Kruskal's algorithm, example



Figure 11: Choose an edge with minimal weight that does not form a cycle.

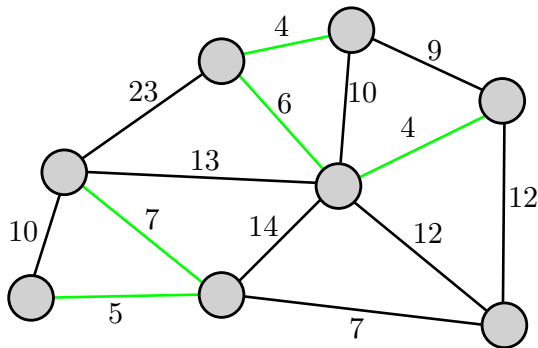# Kruskal's algorithm, example



Figure 12: Add this edge to the tree.
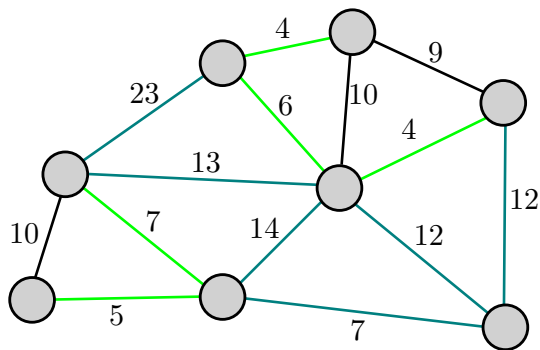
# Kruskal's algorithm, example



Figure 13: Choose an edge with minimal weight that does not form a cycle.

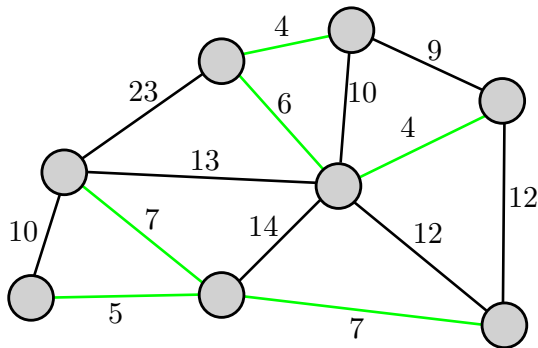# Kruskal's algorithm, example



Figure 14: Add this edge to the tree.
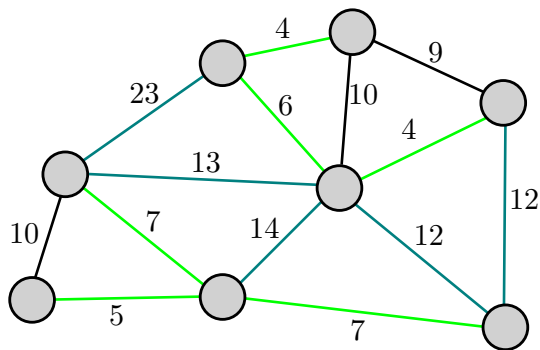
# Kruskal's algorithm, example



Figure 15: Choose an edge with minimal weight that does not form a cycle.

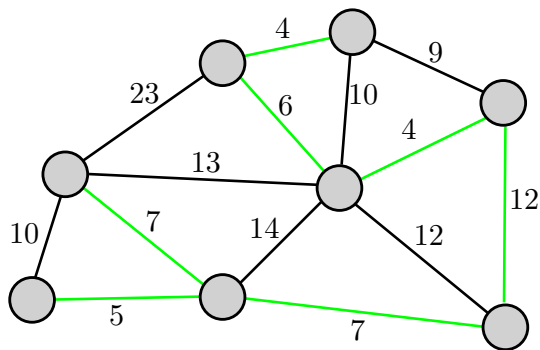# Kruskal's algorithm, example



Figure 16: Add this edge to the tree.
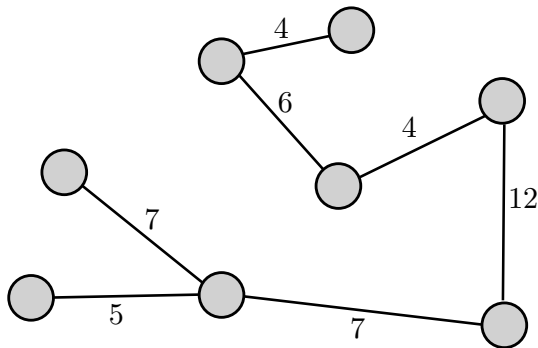
# Kruskal's algorithm, example



Figure 17: The tree is spanning. The algorithm terminates.

# Implementing Kruskal's algorithm

- Kruskal's algorithm can be shown to run in $O(E \log V)$ time.
- By pre-sorting the edges by weight, the step "Choose such an edge with minimal weight" can be performed in constant time.
- To keep track of which vertices are in which components, a *disjoint-set data structure* can be used. This data structure allows efficient implementation of the following operations:
    - *Find:* Determine which subset a particular element is in. (Or determining if two elements are in the same subset).
    - *Union:* Merge two subsets into a single subset.

# Prim's algorithm

## Prim's algorithm

Set $V_{new} = \{v\}$, where $v$ is an arbitrary vertex in $V$.
Set $E_{new} = \emptyset$.
**while** $V_{new} \neq V$ **do**
    *Choose an edge $e_{p,q}$ with minimal weight such that $p$ is in $V_{new}$ and $q$ is not.*
    *Add $q$ to $V_{new}$ and $e_{p,q}$ to $E_{new}$.*
**end**

- At the termination of the algorithm, $(V, E_{new})$ is a MST on $G$.
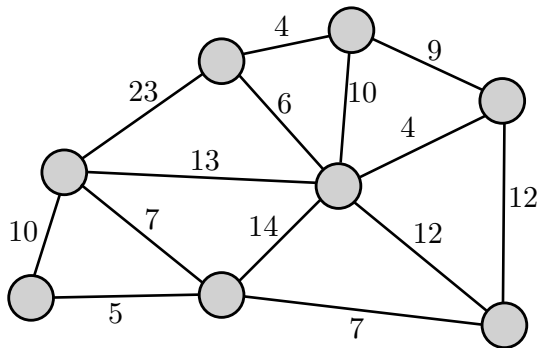
# Prim's algorithm, example



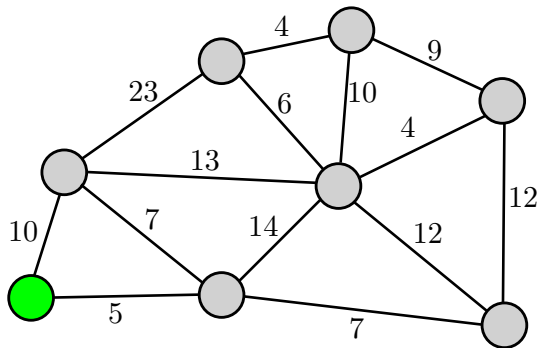Figure 18: An edge weighted graph.

# Prim's algorithm, example



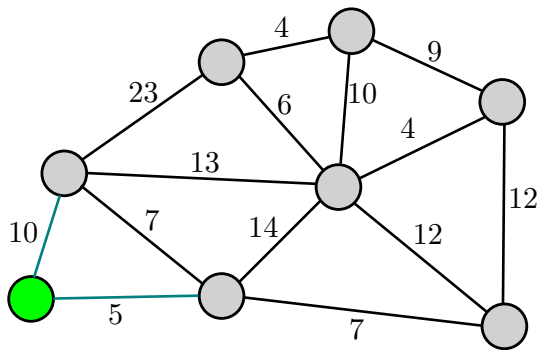Figure 19: Start by adding an arbitrary vertex to $V_{new}$.

# Prim's algorithm, example



Figure 20: Choose a minimal edge $e_{p,q}$ with such that $p$ is in $V_{new}$ and $q$ is not.

# Prim's algorithm, example



Figure 21: Add $q$ to $V_{new}$ and $e_{p,q}$ to $E_{new}$.

# Prim's algorithm, example



Figure 22: Choose a minimal edge $e_{p,q}$ with such that $p$ is in $V_{new}$ and $q$ is not.

Centre for Image Analysis
Uppsala University

UPPSALA
UNIVERSITET

# Prim's algorithm, example



Figure 23: Add $q$ to $V_{new}$ and $e_{p,q}$ to $E_{new}$.

# Prim's algorithm, example



Figure 24: Choose a minimal edge $e_{p,q}$ with such that $p$ is in $V_{new}$ and $q$ is not.
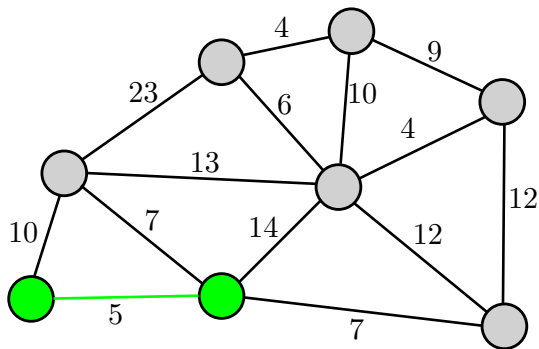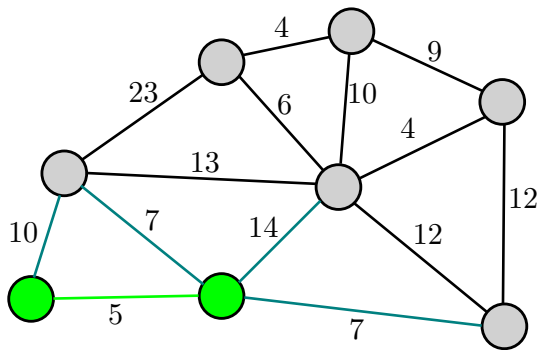
# Prim's algorithm, example



Figure 25: Add $q$ to $V_{new}$ and $e_{p,q}$ to $E_{new}$.

# Prim's algorithm, example



Figure 26: Choose a minimal edge $e_{p,q}$ with such that $p$ is in $V_{new}$ and $q$ is not.
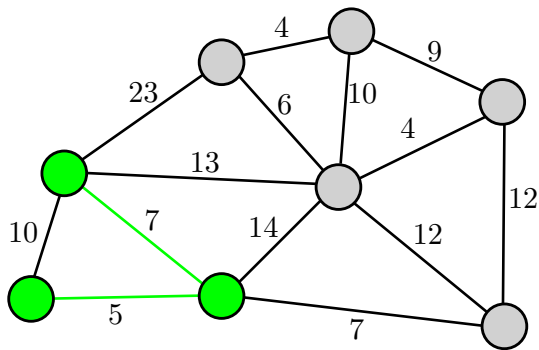
# Prim's algorithm, example



Figure 27: Add $q$ to $V_{new}$ and $e_{p,q}$ to $E_{new}$.
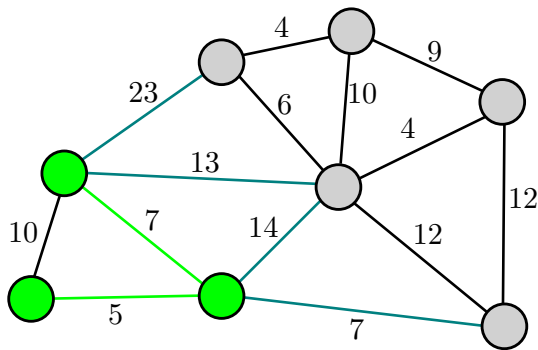
# Prim's algorithm, example



Figure 28: Choose a minimal edge $e_{p,q}$ with such that $p$ is in $V_{new}$ and $q$ is not.

# Prim's algorithm, example



Figure 29: Add $q$ to $V_{new}$ and $e_{p,q}$ to $E_{new}$.

# Prim's algorithm, example



Figure 30: Choose a minimal edge $e_{p,q}$ with such that $p$ is in $V_{new}$ and $q$ is not.
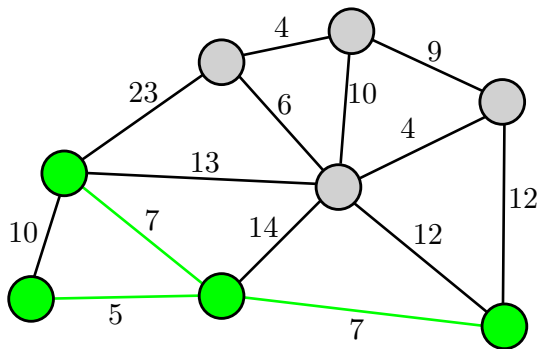
# Prim's algorithm, example
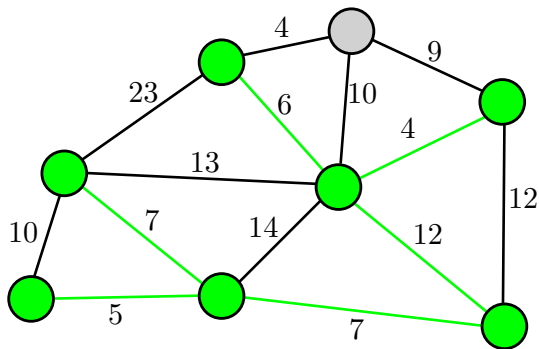


Figure 31: Add $q$ to $V_{new}$ and $e_{p,q}$ to $E_{new}$.
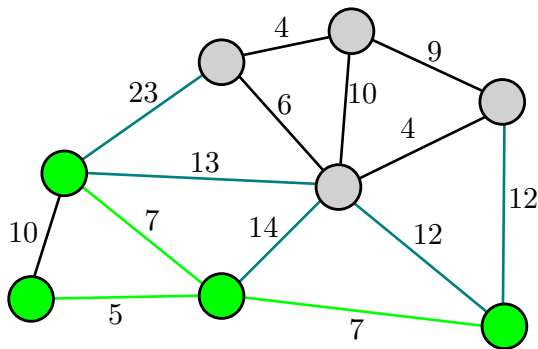
# Prim's algorithm, example
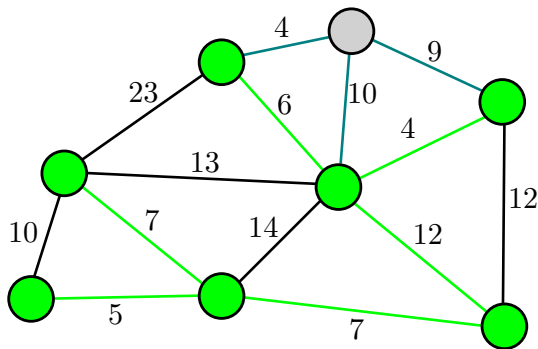


Figure 32: Choose a minimal edge $e_{p,q}$ with such that $p$ is in $V_{new}$ and $q$ is not.
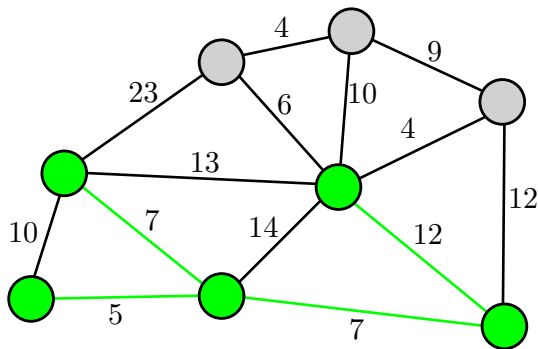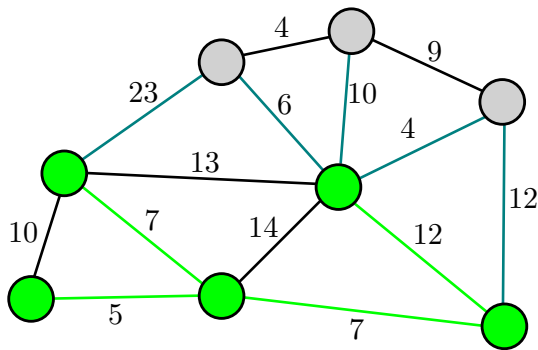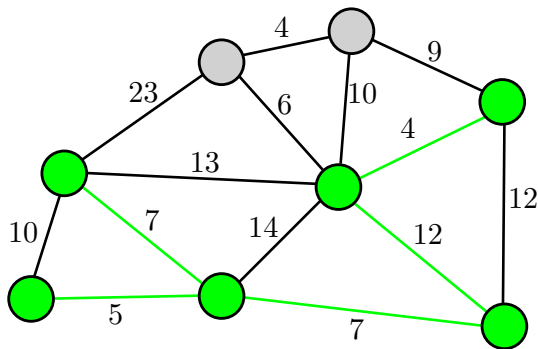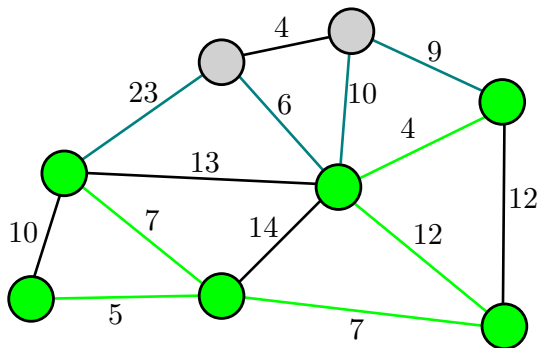
# Prim's algorithm, example
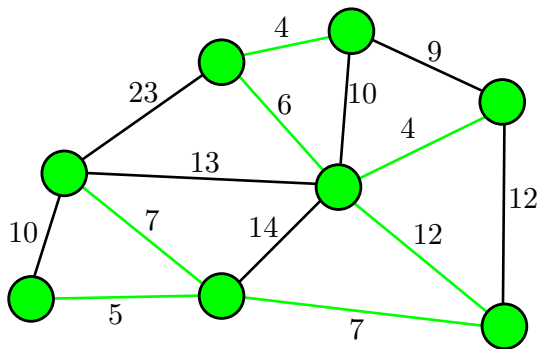


Figure 33: Add $q$ to $V_{new}$ and $e_{p,q}$ to $E_{new}$.

# Prim's algorithm, example
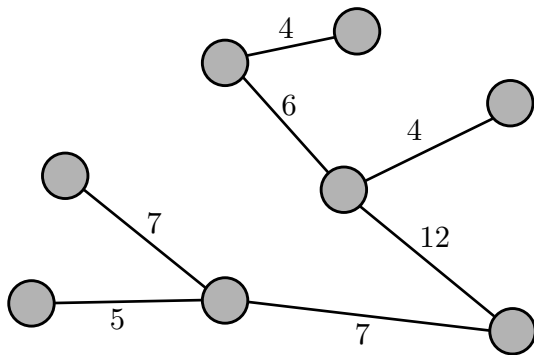


Figure 34: $V_{new} = V$. The algorithm terminates.

# Implementing Prim's algorithm

- The edges are not neccesarily visited in increasing order, so we can't pre-sort the edges.
- Instead, we can use some variant of a *priority queue* to efficiently find the next edge with minimum weight.
- With such an implementation, Prim's algorithm can be shown to run in $O(E \log V)$.

UPPSALA
UNIVERSITET

# Spanning forests relative to seeds

## Definition, spanning forest

Let $G$ be a connected, undirected graph, and let $S \subseteq V$ be a set of *seedpoints*. Let $T$ be a subgraph of $G$ such that

- $T$ is a forest.
- $V(T) = V(G)$.
- Each connected component of $T$ contains exactly one seedpoint.

Then $T$ is a *spanning forest* of $G$, relative to $S$.

# Minimum spanning forests

- A spanning forest $T$ of $G$ is a *minimum spanning forest* (MSF) if the sum of the edge weights is smaller than for any other spanning forest relative to $S$.
- We can use Prim's or Kruskal's algorithms, with slight modifications, to compute MSFs.

# Minimum spanning forests and segmentation

- A MSF partitions a graph into a number of components, each containing exactly one seed-point.
- We will now examine how this can be used for seeded segmentation.
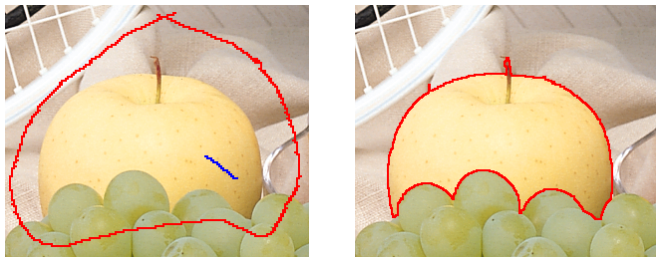


Figure 35: Left: Seed-points representing background (red) and object (blue). Right: Segmentation by MSFs.

# But each seedpoint defines a connected component?
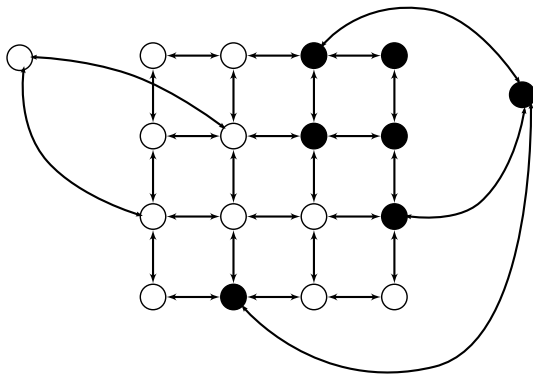


Figure 36: A pixel adjacency graph with "extra" vertices, corresponding to label categories.

# MSF cuts, global optimality

- For any spanning forest $T$ on $G$, we define a *induced cut $C$* as follows:

$$C(T) = \{e_{p,q} \in E \mid p \underset{T}{\not\sim} q\} . \tag{3}$$

- For any cut $C$, we define the *weight* of a cut as

$$\min_{e \in S}(W(e)) . \tag{4}$$

- If $S$ is a cut induced by a MSF, then the weight of $S$ is greater than or equal to the cost of *any* other cut that separates the seedpoints [1].

# Properties of MSF cuts

Contrast invariance

- The MSF computations depend on the relative ordering of the edge weights, but not on the absolute weight values.
- Thus, the segmentation result is invariant under strictly monotonic transformations of the edge weights. (A transformation that preserves the order)

# Properties of MSF cuts

Seed-relative robustness.

- The *core*, or *robustness region*, of a seedpoint is the region (set of vertices) where the seed can be moved without altering the segmentation result.
- For MSF-cuts, the core of each seedpoint can be determined exactly, and is usually large. [2]

UPPSALA
UNIVERSITET

# MSF cuts and Watersheds

There is a strong relation between segmentation by MSFs and the Watershed approach to segmentation:

- J. Cousty et al., *Watershed cuts: minimum spanning forests, and the drop of water principle*. IEEE PAMI, 31(8), 2009.
- J. Cousty et al., *Watershed cuts: Thinnings, shortest path forests, and topological watersheds*. IEEE PAMI, 32(5), 2010.

# Part 3: Shortest path forests

# Shortest paths on graphs

- Let $G$ be a connected, undirected, edge weighted graph. We define the *length* $f(\pi)$ of a path $\pi$ on $G$ as

$$f(\pi) = \sum_{i=1}^{k-1} w(e_{v_i, v_{i+1}}) \ . \tag{5}$$

- For each pair of vertices $v, w$, there exists one or more paths in $G$ that start at $v$ and end at $w$. Among these paths, there is at least one path for which the length is minimal.

- Formally, a path $\pi$ is a *shortest path* if $f(\pi) \leq f(\tau)$ for any other path $\tau$ with $org(\tau) = org(\pi)$ and $dst(\tau) = dst(\pi)$.

# Shortest paths on graphs

- The length of the shortest path between two vertices provides a notion of *distance*, or *degree of connectedness*, between pairs of vertices in the graph.
- Again, we have a global optimization problem: Among all paths between a pair of vertices, we seek one that has minimum length. Fortunately, there are efficient algorithms that solve this problem.
- Given a set $S \subseteq V$ of seed-points, it is in fact possible to simultaneously compute minimal cost paths from $S$ to all other vertices in $V$. The output of this computation is a *shortest path forest*.

# Shortest paths on graphs

- In general, the shortest path between two vertices is not unique. The set of shortest paths between two image elements $p$ and $q$ is denoted $\pi_{min}(p, q)$.
- For two sets $A \subseteq V$ and $B \subseteq V$, $\pi$ is a path between $A$ and $B$ if $org(\pi) \in A$ and $dst(\pi) \in B$. If $f(\pi) \leq f(\tau)$ for any other path $\tau$ between $A$ and $B$, then $\pi$ is a shortest path between $A$ and $B$. The set of shortest paths between $A$ and $B$ is denoted $\pi_{min}(A, B)$.

# Predecessor maps

## Predecessor maps, definition

A *predecessor map* is a mapping $P$ that assigns to each vertex $v \in V$ either an element $w \in \mathcal{N}(v)$, or $\emptyset$.

For any $v \in V$, a predecessor map $P$ defines a path $P^*(p)$ recursively. We denote by $P^0(v)$ the first element of $P^*(v)$.

Centre for Image Analysis
Uppsala University

UPPSALA
UNIVERSITET

# Spanning forests as predecessor maps

## Spanning, definition

A *spanning forest* is a predecessor map that contains no cycles, i.e., $|P^*(v)|$ is finite for all $v \in V$. If $P^*(v) = \emptyset$, then $v$ is a *root* of $P$.

## Shortest path forests

Let $S \subseteq V$. If $P$ is a spanning forest such that $P^*(v) \in \pi_{min}(v, S)$ for all vertices $v \in V$, then we say that $P$ is an *shortest path forest* with respect to $S$.

# Computing shortest path forests

- In 1956, Dijkstra [4] for computing shortest path forests.
- The algorithm is based on the observation that if $\pi = \pi_1 \cdot \pi_2$ is a shortest path between $org(\pi)$ and $dst(\pi)$, then $\pi_1$ and $\pi_2$ must also be shortest paths between their respective endpoints.
- Thus, we can recursively reduce the problem to a set of "smaller" subproblems.

# Dijkstra's algorithm

Input: A graph $G = (V, E)$ and a set $S \subseteq V$ of seed-points.
Auxillary: Two set of vertices $\mathcal{F}$ and $\mathcal{Q}$ whose union is $V$.
Set F$\leftarrow \emptyset$,Q$\leftarrow V$.
For all $v \in V$, set $P(v) \leftarrow \emptyset$.
**while** $\mathcal{Q} \neq \emptyset$ **do**

    Remove from Q a vertex $v$ such that $f(P * (v))$ is minimum, and add it to $\mathcal{F}$.

    **foreach** $w \in \mathcal{N}(w)$ **do**

        | If $f(P^*(w) \cdot \langle w, v \rangle < f(P^*(v)))$, then set $P(w) \leftarrow v$.

    **end**

**end**

# Dijkstra's algorithm, example
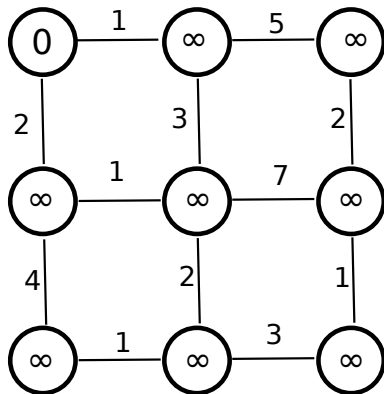


Figure 37: Dijkstra's algorithm.

# Dijkstra's algorithm, example
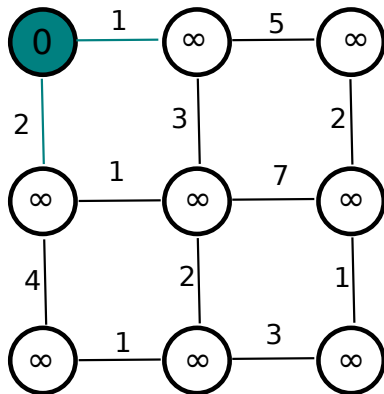


Figure 38: Dijkstra's algorithm.

# Dijkstra's algorithm, example
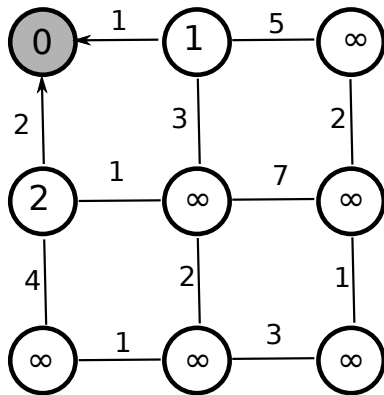


Figure 39: Dijkstra's algorithm.

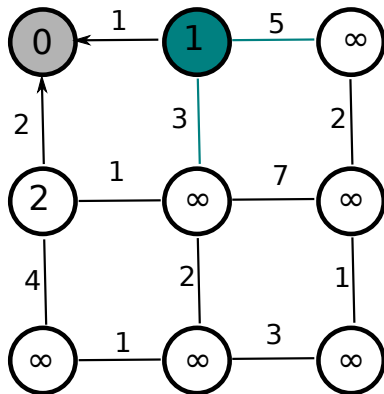# Dijkstra's algorithm, example



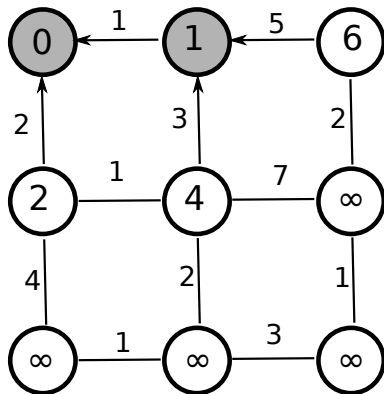Figure 40: Dijkstra's algorithm.

# Dijkstra's algorithm, example



Figure 41: Dijkstra's algorithm.
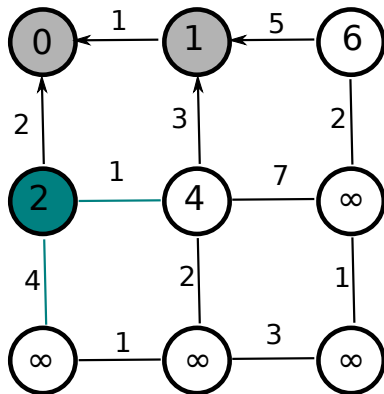
# Dijkstra's algorithm, example



Figure 42: Dijkstra's algorithm.
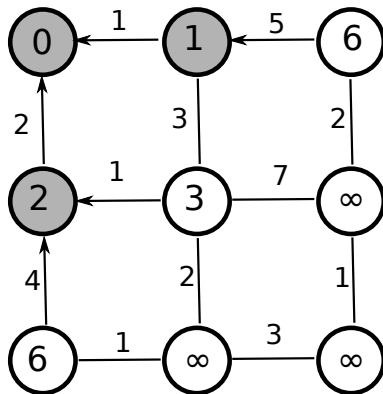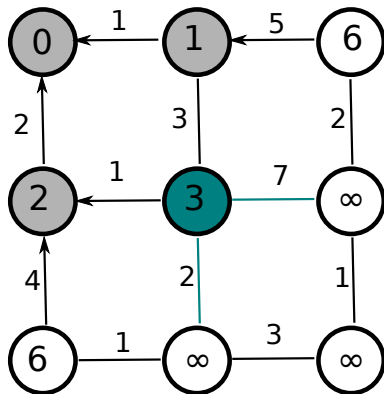
# Dijkstra's algorithm, example



Figure 43: Dijkstra's algorithm.

# Dijkstra's algorithm, example



Figure 44: Dijkstra's algorithm.
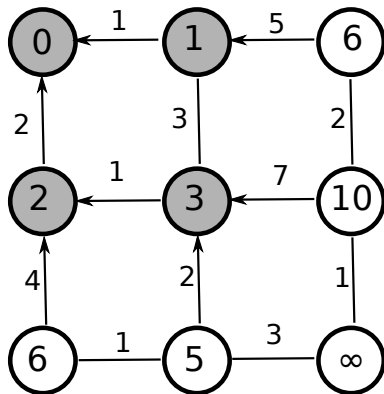
# Dijkstra's algorithm, example



Figure 45: Dijkstra's algorithm.
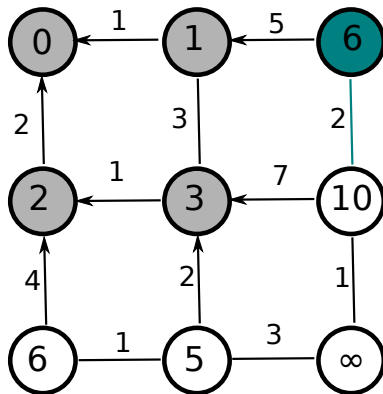
# Dijkstra's algorithm, example



Figure 46: Dijkstra's algorithm.

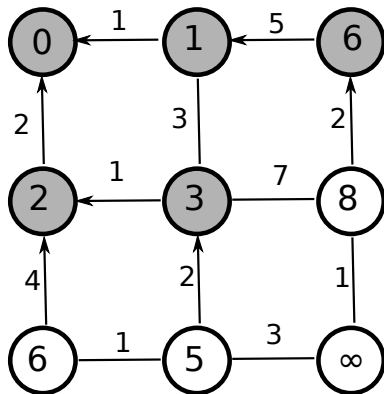# Dijkstra's algorithm, example



Figure 47: Dijkstra's algorithm.

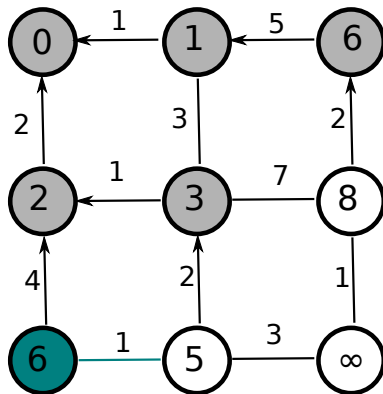# Dijkstra's algorithm, example



Figure 48: Dijkstra's algorithm.

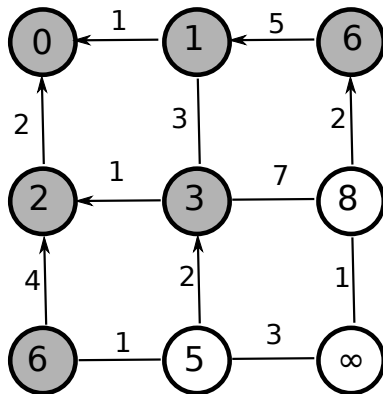# Dijkstra's algorithm, example



Figure 49: Dijkstra's algorithm.

# Dijkstra's algorithm, example



Figure 50: Dijkstra's algorithm.

# Dijkstra's algorithm, example



Figure 51: Dijkstra's algorithm.

# Dijkstra's algorithm, example



Figure 52: Dijkstra's algorithm.

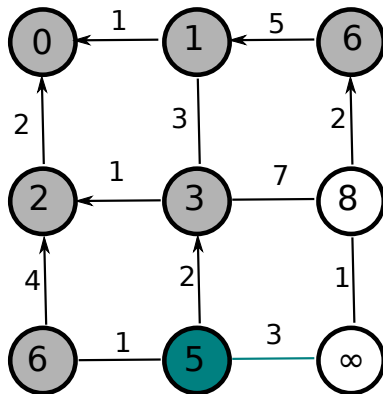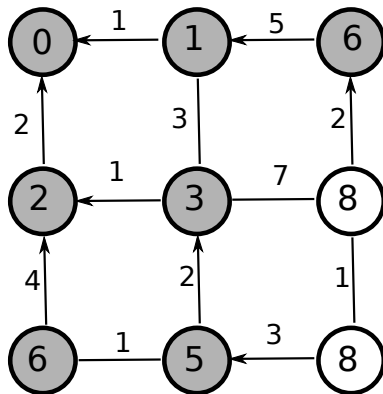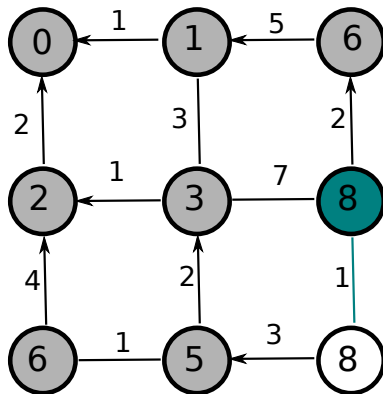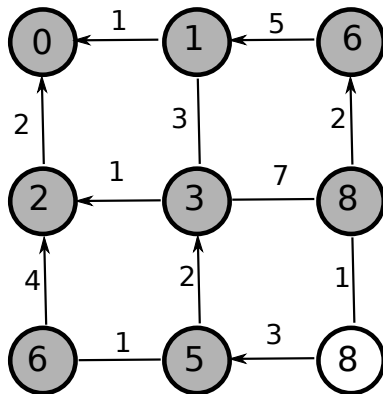# Dijkstra's algorithm, example



Figure 53: Dijkstra's algorithm.

# Dijkstra's algorithm, example
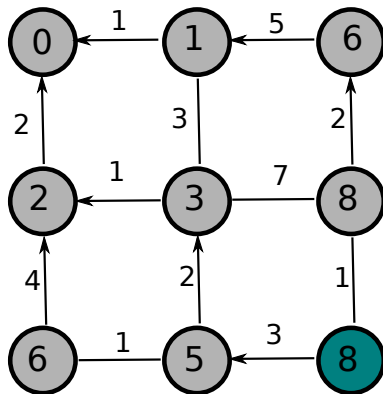


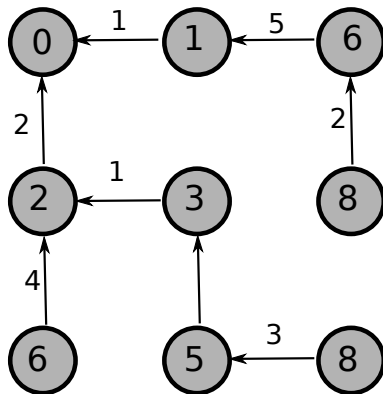Figure 54: Dijkstra's algorithm.

# Dijkstra's algorithm, example



Figure 55: Dijkstra's algorithm.

# Implementing Dijkstra's algorithm

- Just like with Prim's algorithm, we can use a *priority queue* to efficiently extract the vertex for which $f(P^*(v))$ is minimum.
- The algorithm can be shown to run in $O(|E| + |V| \log |V|)$. (For some types of graphs, we can do better)

# Live-wire segmentation

- The perhaps most straightforward way of utilizing shortest cost path calculations in image segmentation is to consider the path itself as a boundary between two regions. This idea is used in the *live-wire* method.

- To segment an object in a 2D image with live-wire, the user selects a point on the object boundary. Dijkstras algorithm is then used to compute shortest paths from this point to all other points in the image.

- As the user moves the pointer through the image, a minimal cost path from the current position to the seed-point the live wire is displayed in real-time.
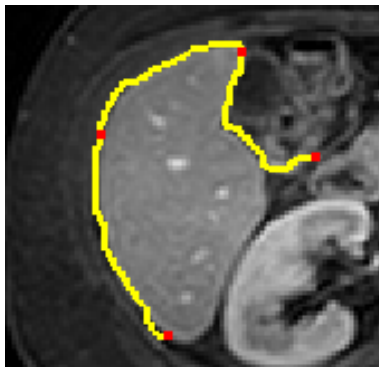
# Live-wire segmentation



Figure 56: Live-wire segmentation.

# Seeded segmentation with shortest paths

- Associate each seed-point with a label, and assign to all other vertices the label of the closest seedpoint as determined by the minimum cost path forest.
- We can modify Dijkstra's algorithm to propagate the labels along with the shortest paths.
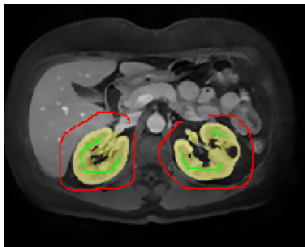


Figure 57: Seeded segmentation with shortest paths.

# Approximating Euclidean distances

- The length of the minimal cost path between two vertices can be interpreted as a "distance" between them.
- On a 2D or 3D regular grid, the cost of the minimal path between two vertices can approximate the Euclidean distance between the corresponding points.
- The quality of this approximation depends on the definition of the graph, and the selection of edge weights [8].

# Approximating Euclidean distances
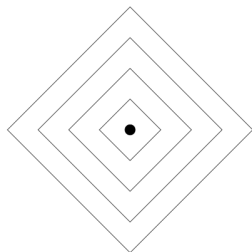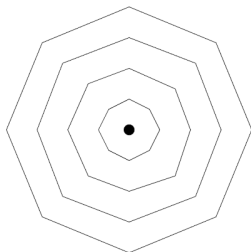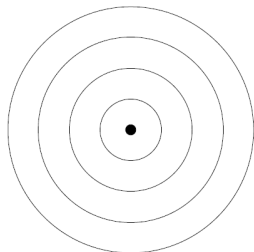


(a) 4 n-system    (b) 8 n-system    (c) 128 n-system

Figure 58: Distances in discrete grids [3]. The weight of each edge is equal to its Euclidean length.

# Distance maps

We can use Dijkstra's algorithm to compute *distance maps*, where every pixel is assigned the cost of the shortest path from the pixel to the nearest seedpoint.



Figure 59: Left: A cameraman. Right: Seeds.

# Distance maps

We can use Dijkstra's algorithm to compute *distance maps*, where every pixel is assigned the cost of the shortest path from the pixel to the nearest seedpoint.
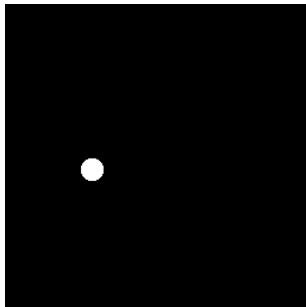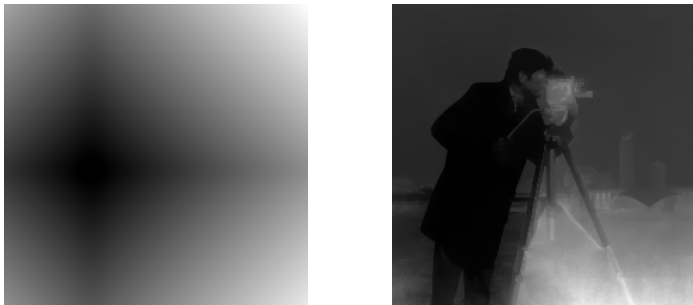


Figure 60: Left: Distance map with all edge weights set to 1. Right: Distance map with edges weighted by absolute local intensity difference.

# Extensions of Dijkstra's algorithm

For now, we have defined the length of a path as the sum of edge weights along the path.

- Are there other path cost functions that could be of interest in image processing?
- If so, what conditions do these functions need to satisfy in order to guarantee the existence of a shortest path forest?
- These questions were investigated by Falcao et al. [5].

# References

[1] C. Allène, J-Y Audibert, M. Couprie, J. Cousty, and R. Keriven.
Some links between min-cuts, optimal spanning forests and watersheds.
In *Proceedings of ISMM*, 2007.

[2] R. Audigier and R. A. Lotufo.
Seed-relative segmentation robustness of watershed and fuzzy connectedness approaches.
In A. X. Falcão and H. Lopes, editors, *Proceedings of the 20th Brazilian Symposium on Computer Graphics and Image Processing*, pages 61–68. IEEE Computer Society, 2007.

[3] Yuri Boykov.
Computing geodesics and minimal surfaces via graph cuts.
In *International Conference on Computer Vision*, pages 26–33, 2003.

[4] Edsger W. Dijkstra.
A note on two problems in connexion with graphs.
*Numerische Mathematik*, 1:269–271, 1959.

[5] Alexandre X. Falcão, Jorge Stolfi, and Robert A. Lotufo.
The image foresting transform: Theory, algorithms, and applications.
*IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(1):19–29, 2004.

[6] Joseph B. Kruskal.
On the shortest spanning subtree of a graph and the traveling salesman problem.
*Proceedings of the American Mathematical Society*, 7(1), 1956.

[7] Robert C. Prim.
Shortest connection networks and some generalizations.
*Bell System Technical Journal*, 36, 1957.

[8] R. Strand.
*Distance Functions and Image Processing on Point-Lattices*.
PhD thesis, Uppsala University, 2008.