Maximum flows and minimal cuts

Filip Malmberg

MSF cuts I

- A Minimum Spanning Forest with respect to a set of seed-points divides a graph into a number of connected components. Thus, it defines a cut on the graph – a MSF cut.
- A cut given by a minimum spanning forest maximizes

$$\min_{e \in S}(w(e)) \tag{1}$$

  among all cuts that separate the seed-points.
- In the above equation, the weight $w(e)$ of an edge represents the *dissimilarity* between the vertices connected by the edge.
- Thus, we are maximizing dissimilarity across the cut.

MSF cuts II

- A Maximum Spanning Forest with respect to a set of seed-points divides a graph into a number of connected components. Thus, it defines a cut on the graph – a MSF cut.
- A cut given by a maximum spanning forest minimizes

$$\max_{e \in S}(w(e)) \tag{2}$$

  among all cuts that separate the seed-points.
- In the above equation, the weight $w(e)$ of an edge represents the *similarity* between the vertices connected by the edge.
- Thus, we are minimizing similarity across the cut.

MSF cuts

- With MSF cuts, we only consider the maximum weight among the edges in the cut.
- There is no penalty for "large" cuts – this can lead to very "jagged" boundaries in noisy images.
- It seems interesting to optimize for a function that considers *all* edges in the cut.
- In this lecture, we will look at a method that computes cuts where the *sum* of the edge weights is minimal.

Minimal graph cuts

- Consider a graph $G = (V, E)$, where we have selected two vertices $s, t \in V$.
- A cut on G is an $s - t$ cut if it separates s from t.
- We want to find an $s - t$ cut C that minimizes

$$\sum_{e \in C} w(e) \, . \tag{3}$$

Minimal $s - t$ cuts

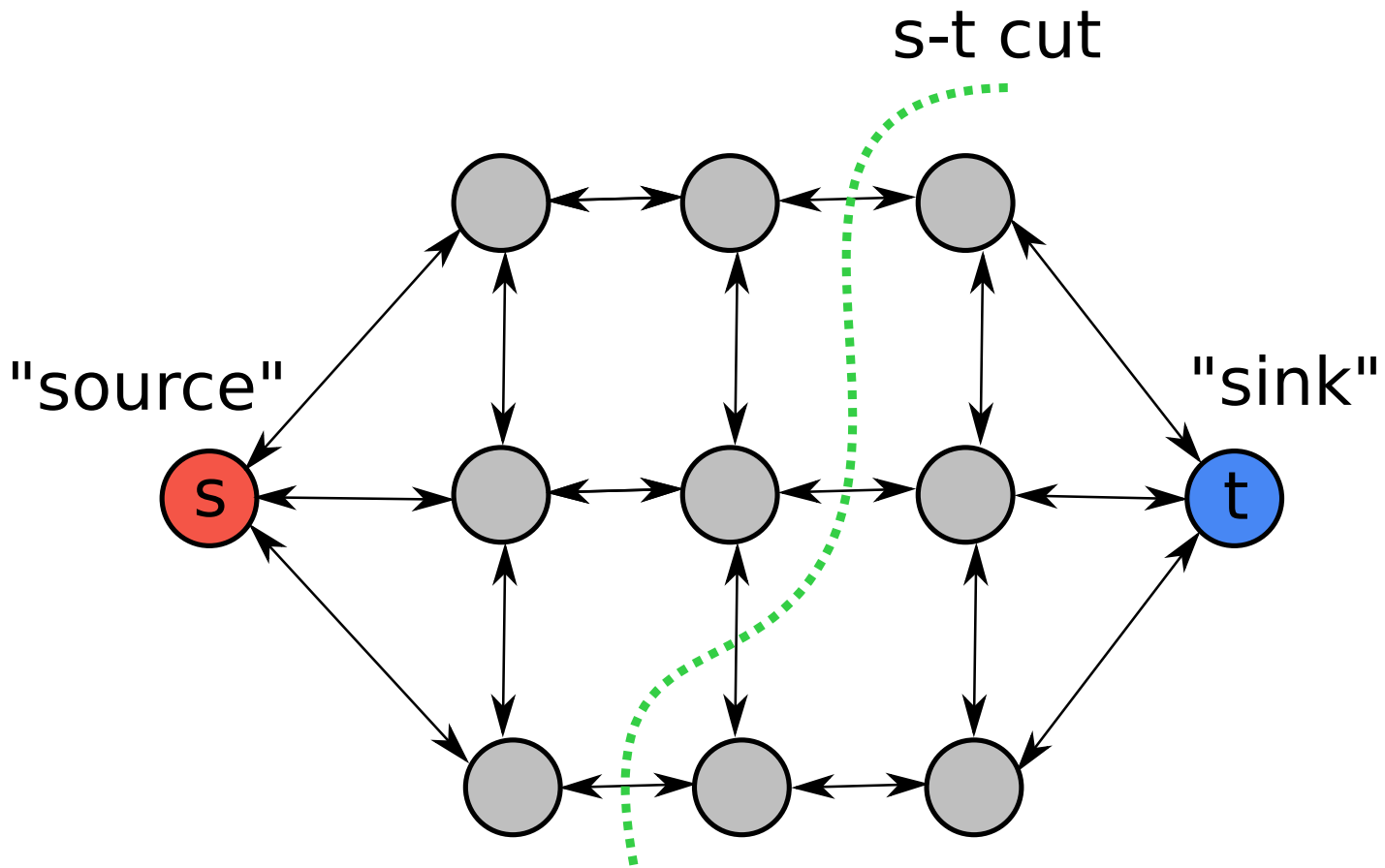


Figure 1: $s - t$ graph cut

P-norms

- The sum and the maximum are special cases of *p-norms*.
- Let $p \geq 1$ be a real number. The *p-norm* of a vector $\mathbf{x}$ is defined as

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \tag{4}$$

- For $p = 1$, this is the sum of the elements of the vector. For $p \to \infty$, it approaches the maximum of the elements.
- If we can solve for the sum, then we can solve for any (finite) p-norm. (We don't really need to care about the p:th root)

Flow network, intuitive notion

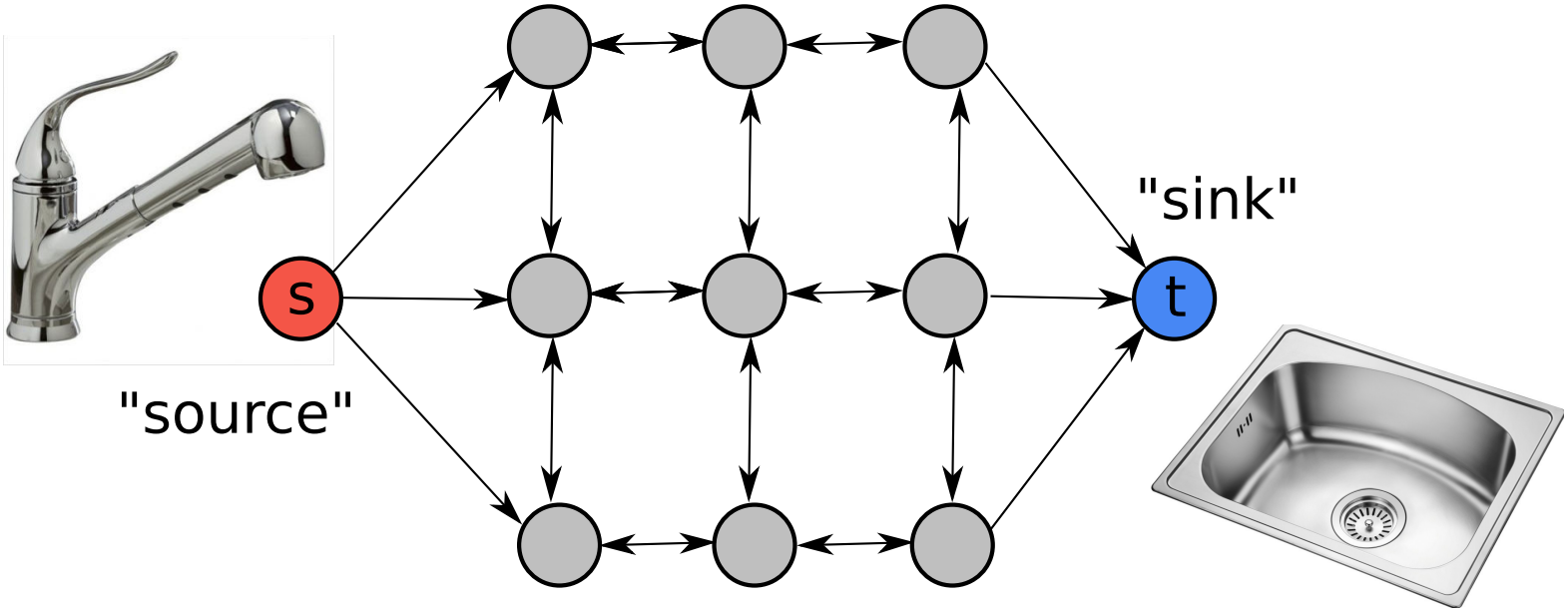


Figure 2: Flow network

Networks

Definition, network

A *network* is a directed graph $G = (V, E)$ where

1. Two vertices $s, t \in V$ are defined as the *source* and *sink* of G, respectively. The source has only outgoing edges and the sink has only incoming edges.
2. a *capacity* function, $c : E \to \mathbb{R}^+$ maps each edge to how much "traffic" it can carry.

Flow

Definition, flow

Given a network $G = (V, E)$, a *(s-t) flow* is a mapping $f : E \to \mathbb{R}^+$, such that:

1. $f(p, q) \leq c(p, q)$ for all $e_{p,q} \in E$. (Capacity constraint)
2. $\sum_{q \in \mathcal{N}(p)} f(p, q) = \sum_{q \in \mathcal{N}(p)} f(q, p)$ for all $p \in V \setminus \{s, t\}$. (Flow conservation)

Maximum flow

- The value, $|f|$, of a flow is the total amoung of flow being sent from the sink to the source, i.e.,

$$|f| = \sum_{p \in \mathcal{N}(s)} f(s, p) \, . \tag{5}$$

- The *maximum flow problem* is to maximize $|f|$, i.e., to route as much flow as possible from s to t.

Ford-Fulkerson theorem

Theorem

The maximum value of a $s-t$ flow on G is equal to the minimum capacity of an $s-t$ cut.

Moreover, a maximum flow on G will *saturate* a set of edges that gives us the minimum cut.

Computing minimum cuts/maximal flows

- According to the Ford-Fulkerson theorem, we can compute minimum s-t cuts by computing maximum flow.
- We will look at one algorithm for computing maximum flows: the Ford-Fulkerson algorithm [5].
- First, we will consider a greedy approach to finding a maximum flow.

Towards the Ford-Fulkerson algorithm

Definition, augmenting path

Let $G = (V, E)$ be a network and let f be a flow on G. A path π in G is called an *augmenting path* if

1. The origin of π is s and the destination of π is t.
2. $f(p, q) < c(p, q)$ for all edges $e_{p,q}$ along the path.

A greedy algorithm

while *There exists an augmenting path in* G **do**
  | Send flow along that path
end

A greedy algorithm

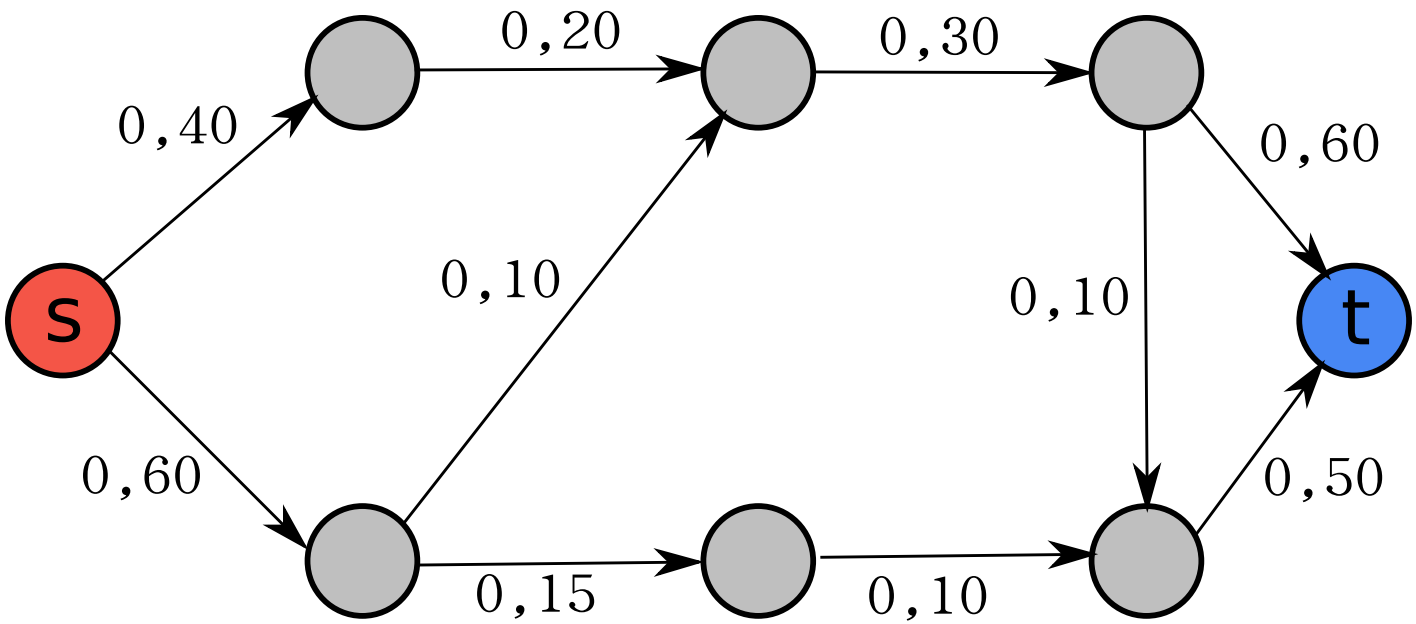


Figure 3: A network with zero flow.

A greedy algorithm

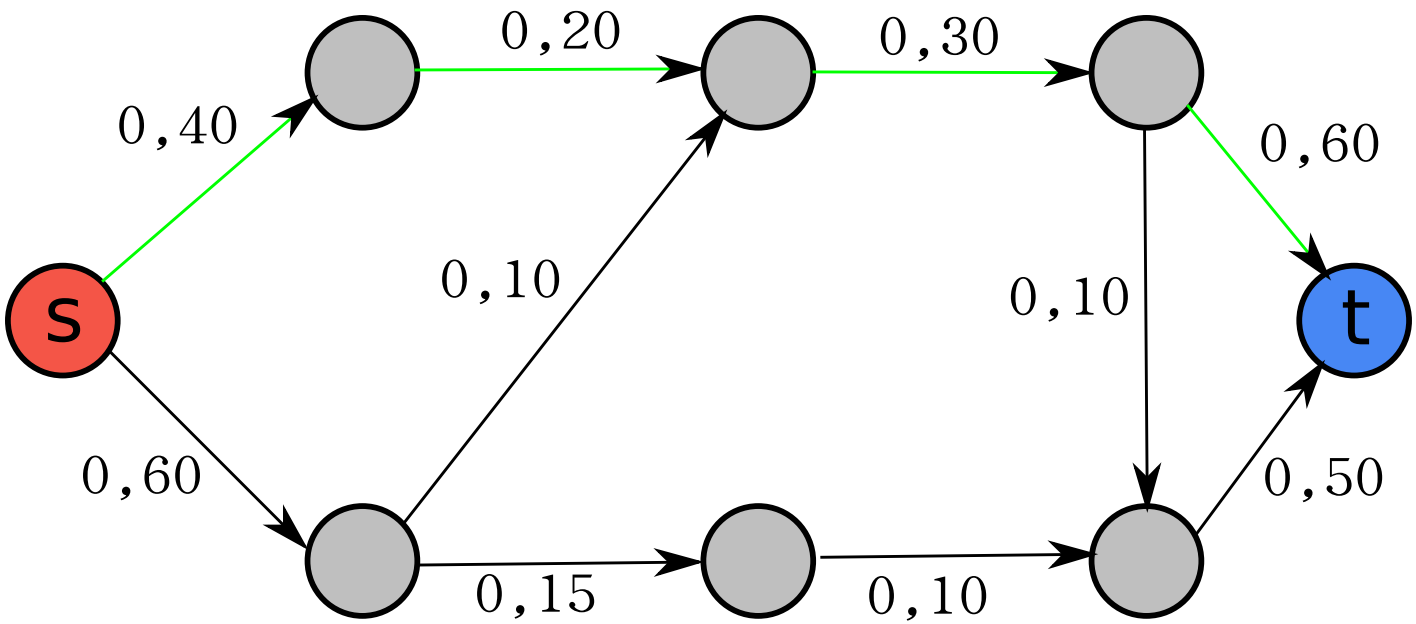


Figure 4: Find an augmenting path.

A greedy algorithm

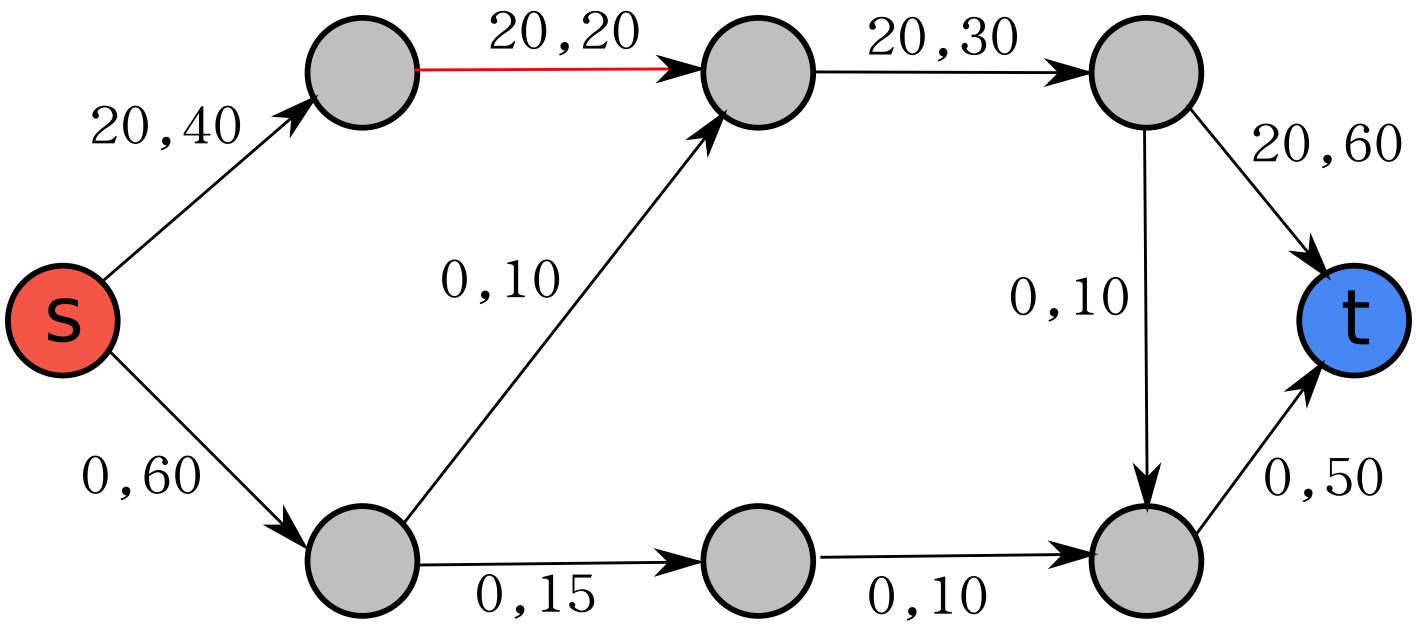


Figure 5: Send flow along the path.

A greedy algorithm

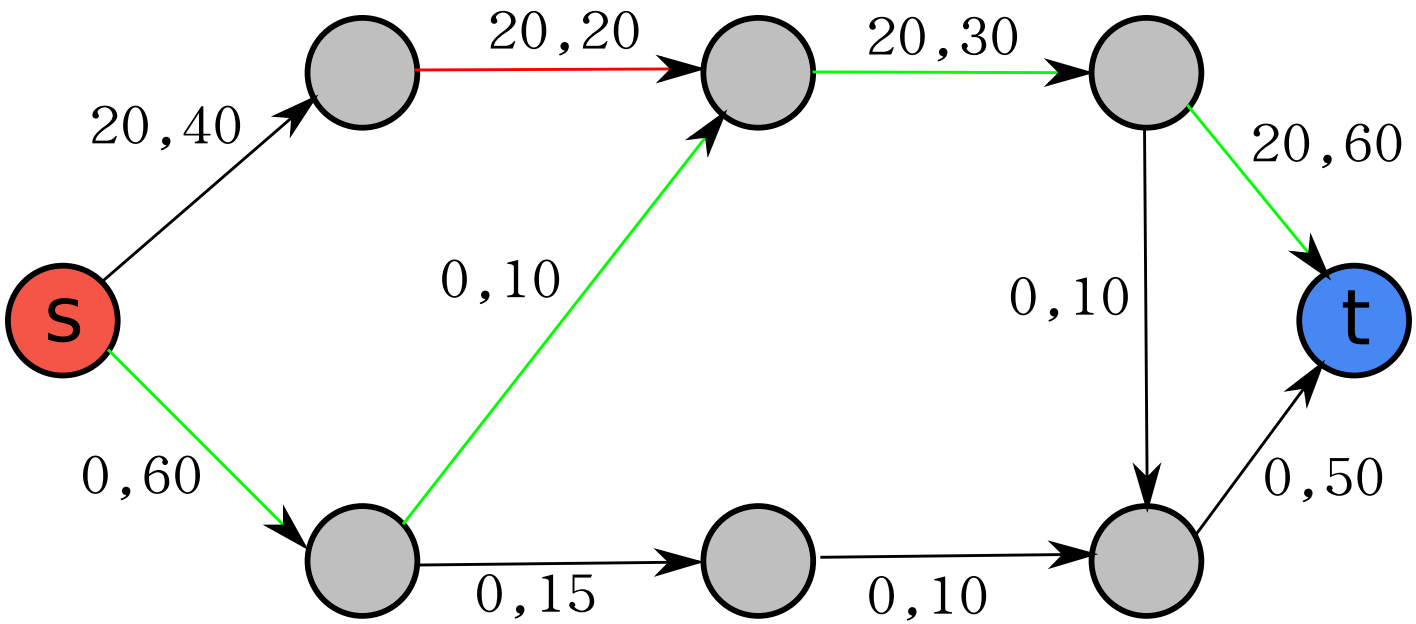


Figure 6: Find an augmenting path.

A greedy algorithm

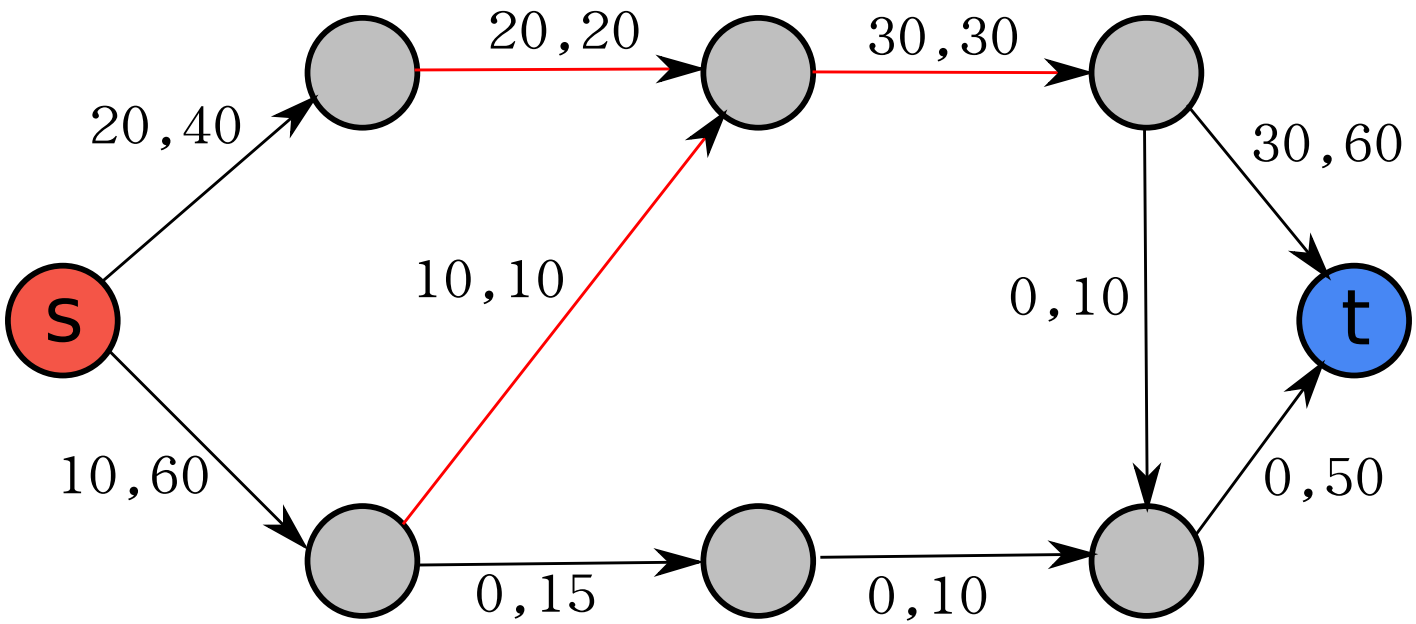


Figure 7: Send flow along the path.

A greedy algorithm

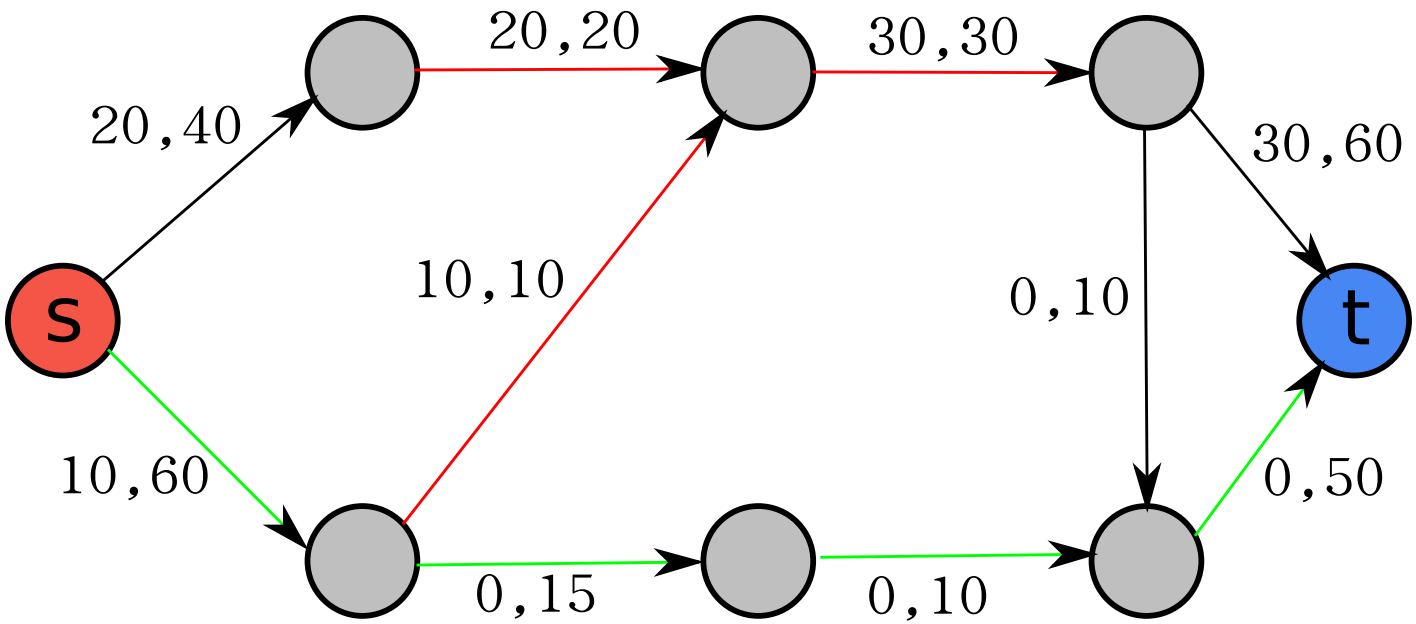


Figure 8: Find an augmenting path.

A greedy algorithm

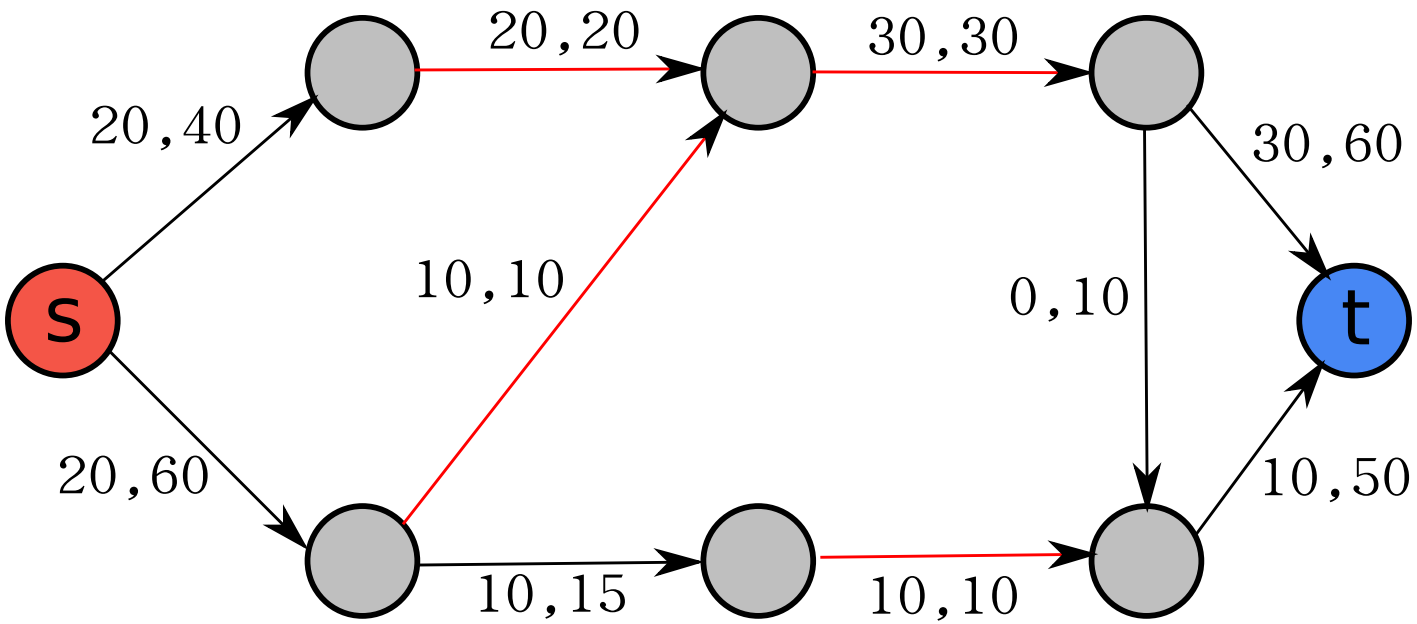


Figure 9: Send flow along the path.

A greedy algorithm

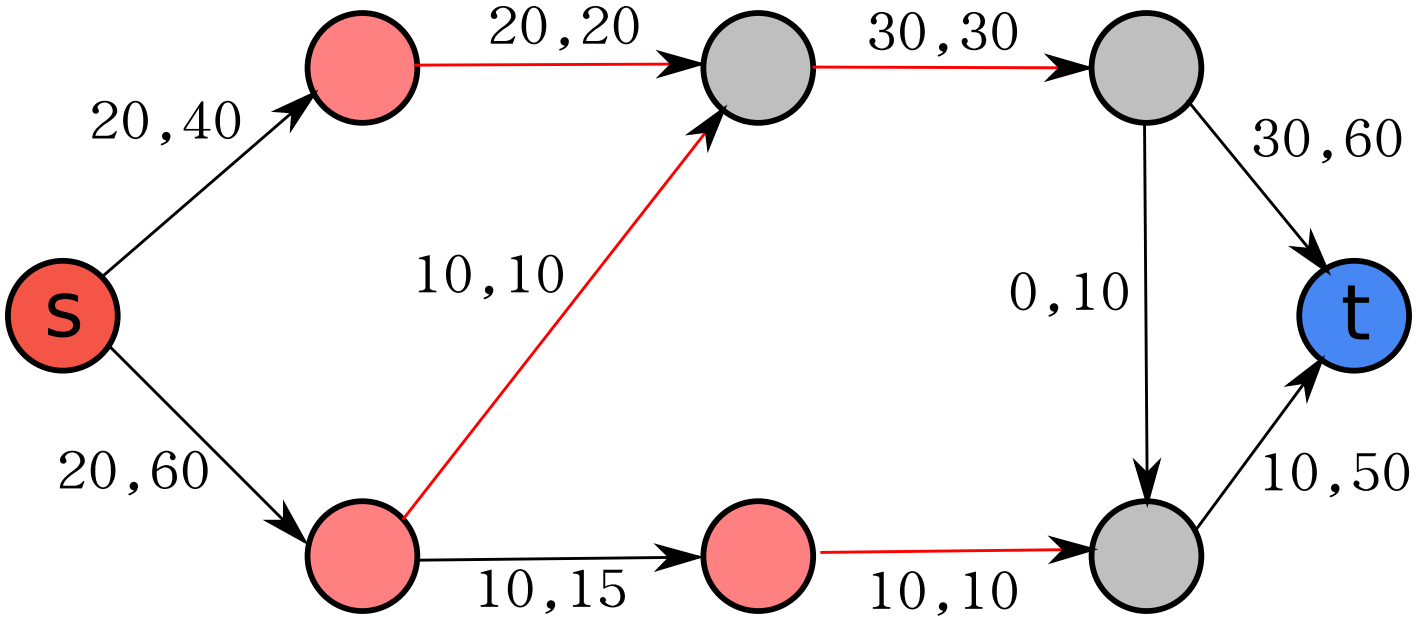


Figure 10: No more augmenting paths can be found. Label all vertices that can be reached via non-saturated edges as "belonging to the source".

A greedy algorithm

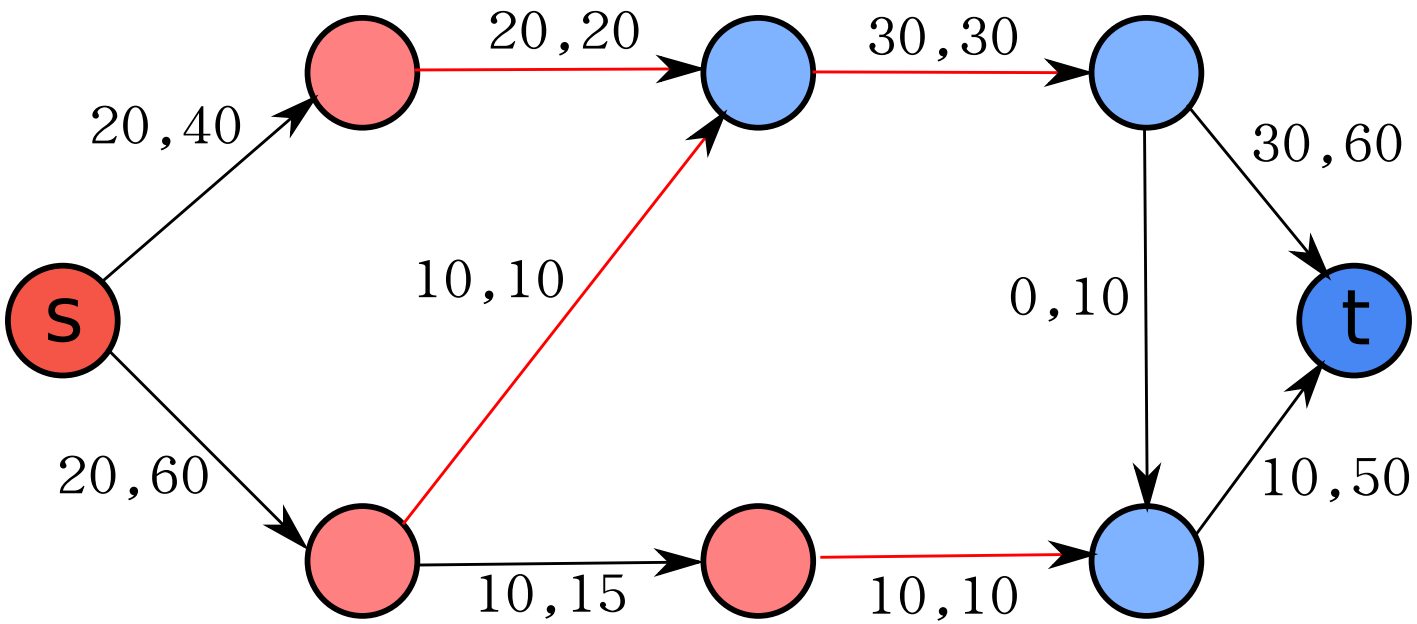


Figure 11: Label all remaining vertices as "belonging to the sink".

A greedy algorithm

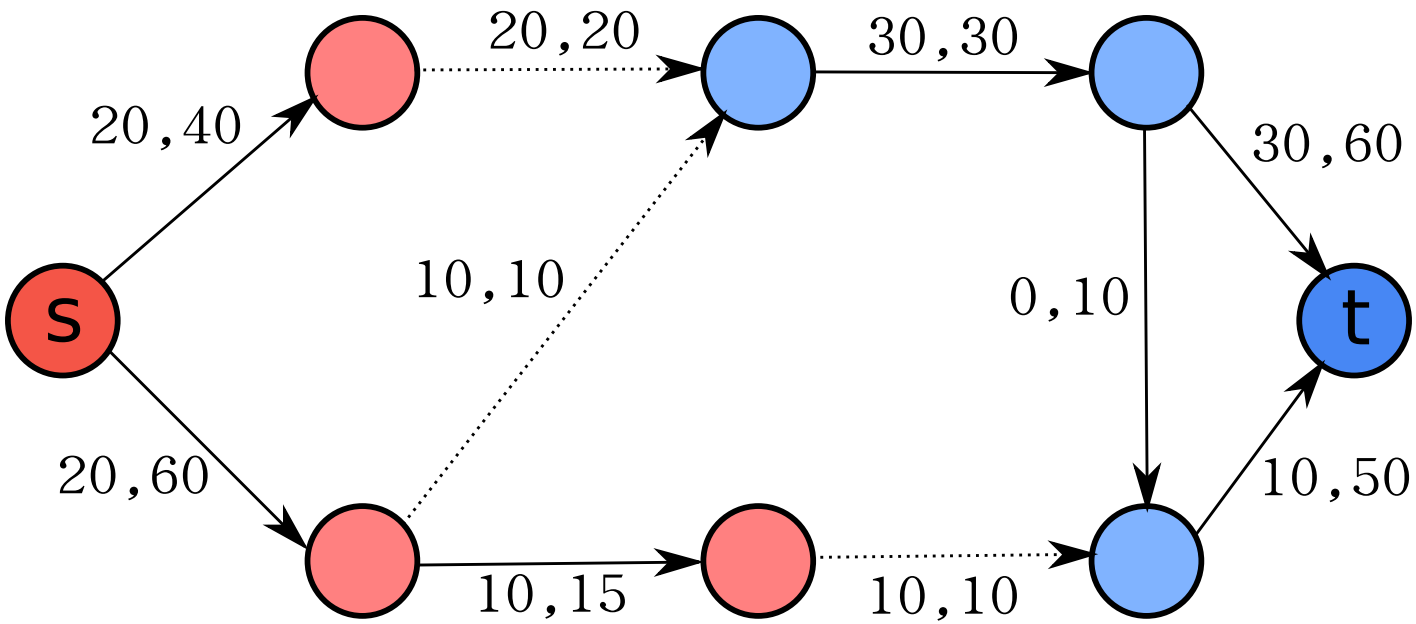
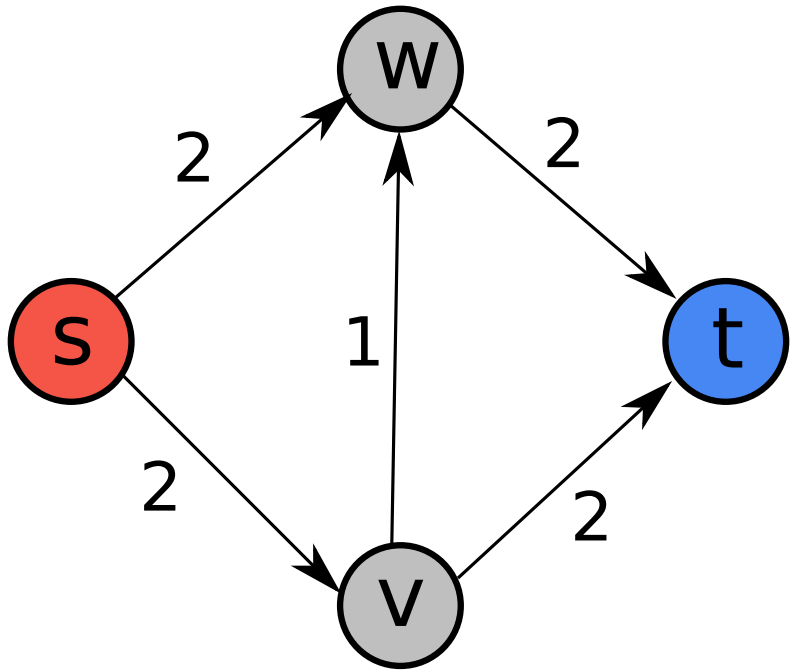


Figure 12: The edges on the boundary of this labeling form a minimum s-t cut.

The greedy algorithm may fail to produce the maximum flow

- The graph below has a unique maximum flow, where the flow on the edge $e_{v,w}$ is zero.
- The greedy algorithm could choose $\{s, v, w, t\}$ as the first augmenting path.

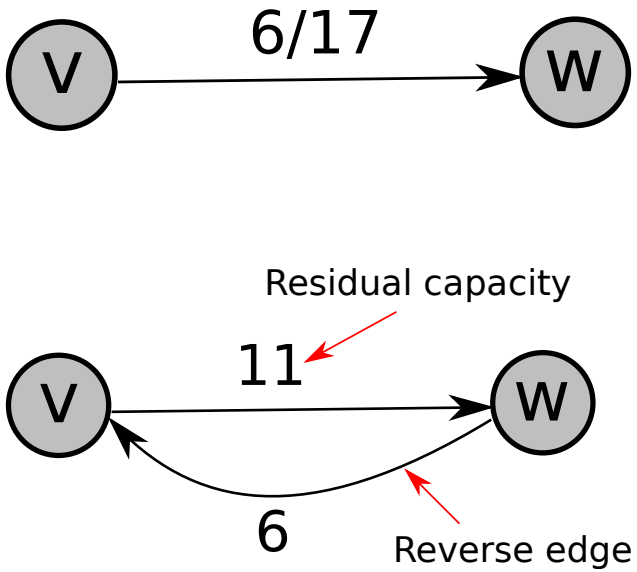


The greedy algorithm may fail to produce the maximum flow

- Once the greedy algorithm increases the flow on an edge, it never decreases it.
- *We need a mechanism to undo bad decisions!*

Residual network

A network with the same vertex set as G. For every original edge e with flow $f(e)$ and capacity $c(e)$, we add an edge with capacity $c(e) - f(e)$ to the residual network. We also add a *reverse* edge, going in the opposite direction of e, with capacity $f(e)$.



Ford-Fulkerson algorithm

while *There exists an augmenting path in the residual graph of* G **do**
  | Augment flow along that path
end

Augmenting flow

The *bottleneck capacity* of an augmenting path is the minimum remaining capacity of any edge along the path. The following algorithm is used to augment flow in the Ford-Fulkerson algorithm:

b $\leftarrow$ bottleneck capacity of an augmenting path π
foreach *edge* e *along* π **do**
  if $e \in E$ **then**
    Set $f(e) \leftarrow f(e) + b$
  end
  else
    Set $f(e) \leftarrow f(e) - b$
  end
end

Ford-Fulkerson algorithm, practicalities

- At each step of the Ford-Fulkerson algorithm, we select an augmenting path.
- The running time of the algorithm depends on the order in which we select the augmenting paths, and on the search strategy we use to find a path.

Edmonds-Karp algorithm

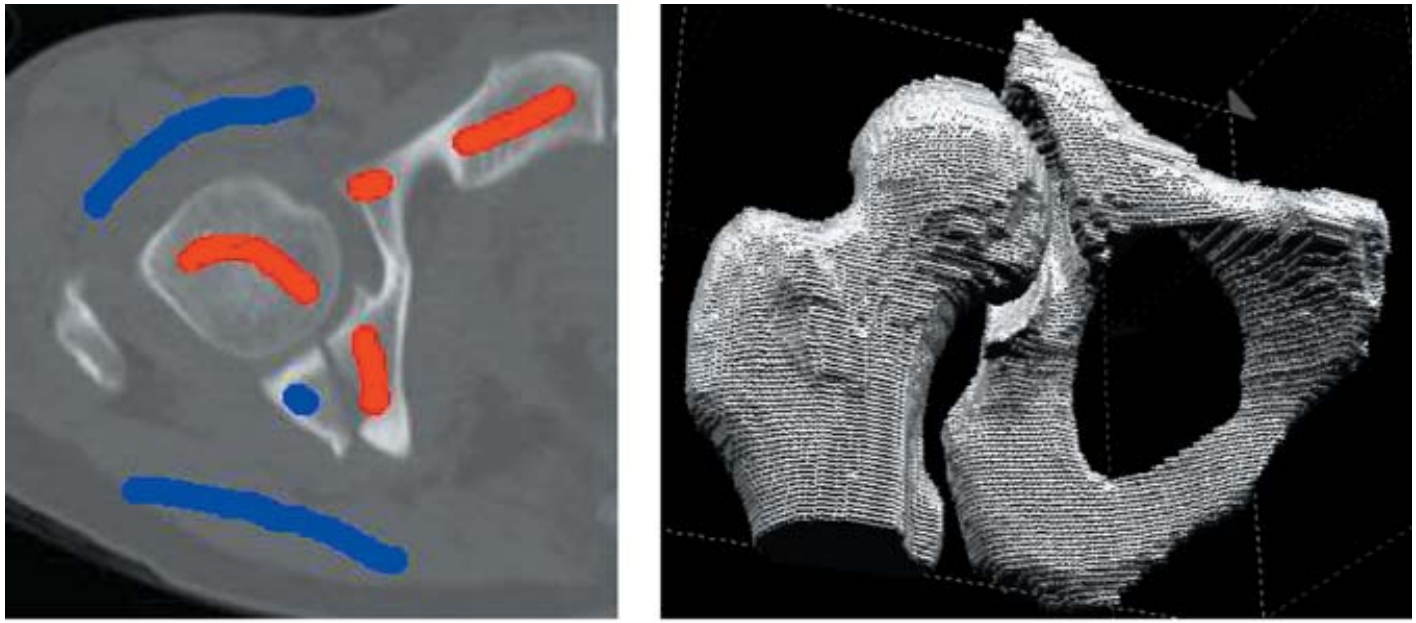
- The Edmonds-Karp algorithm is a specialization of the Ford-Fulkerson algorithm. [3]
- At each step, the algorithm selects a *shortest* augmenting path (where the length of a path is the number of vertices in the path).
- The algorithm can be shown to run in $O(|V||E|^2)$.

Boykov-Kolmogorov algorithm

- Another specialization of the Ford-Fulkerson algorithm, tuned for the types of graphs commonly occuring in image processing [1].
- The basic idea of the algorithm is to maintain two search trees, one from the source and one from the sink. These trees are updated during the execution of the algorithm, so we do not need to perform the search for an augmenting path from scratch.
- The theoretical running time is worse then for the Edmonds-Karp algorithm, but it has been shown to be faster in many practical scenarios.
- An implementation in C is available: http://pub.ist.ac.at/~vnk/software.html

Interactive seeded segmentation

Hard constraints can be implemented with infinity cost terminal links.



(a) A slice with seeds (b) 3D object

Figure 13: Segmentation of bones in a CT volume using minimal graph cuts.

Metrication errors

- We can think of the cost of a cut as the area of a surface separating the two regions.
- On a regular 2D or 3D grid, we will see *metrication errors* – signs of the discrete nature of the graph representation.

Comparison, discrete distance transforms

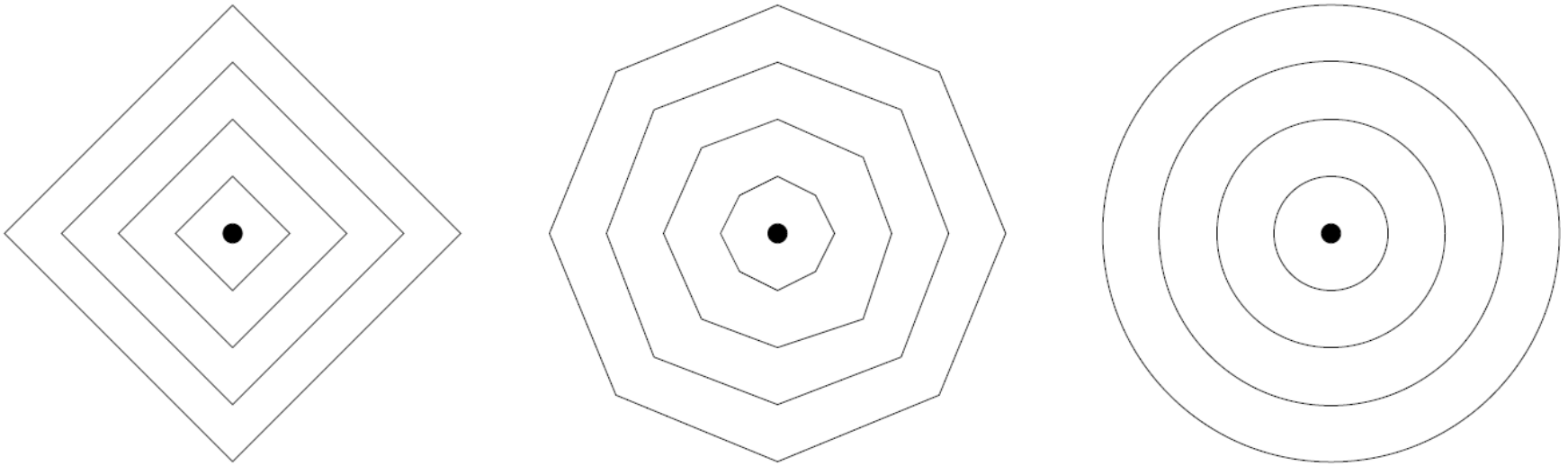


Figure 14: Distances in discrete grids [2]. The weight of each edge is equal to its Euclidean length.

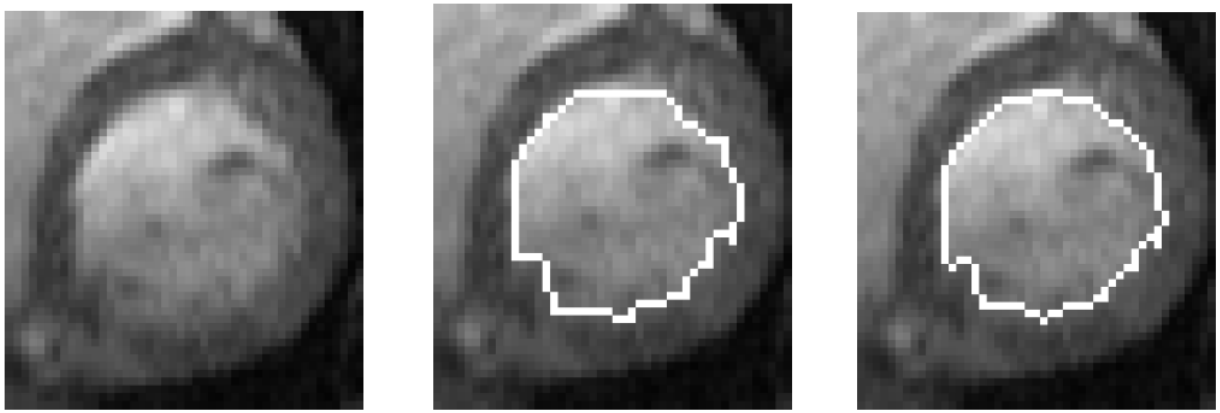
Reducing metrication errors

- Just like in the distance transform example, we can reduce metrication errors by using a "larger" neighborhood system.
- In fact, it is possible to construct a graph such that the weight of the cut is arbitrarily close to the lenght (area) of the corresponding contours (surfaces) for any Riemannian metric [2].

Reducing metrication errors

(a) Original data (b) 4 n-system (c) 8 n-system

image restoration experiments on 2D data



(d) Original data (e) 6 n-system (f) 26 n-system

object extraction experiments on 3D data

Shrinking bias

- Since we are minimizing the sum of the edge weights in the cut, we implicitly favour "small" cuts. This may or may not be what we want.
- To avoid this issue, some authors [6, 4] have considered "normalized cuts" that minimize

$$\frac{\sum_{e \in C} w(e)}{|C|} \,. \tag{6}$$

Graph cuts as a general optimization tool

- The ability to optimize cost functions of the form discussed here has applications to many image processing tasks other than segmentation.
- Examples:
  - Filtering (Labels are intensities).
  - Stereo matching (Labels are disparities or depths).

Image restoration

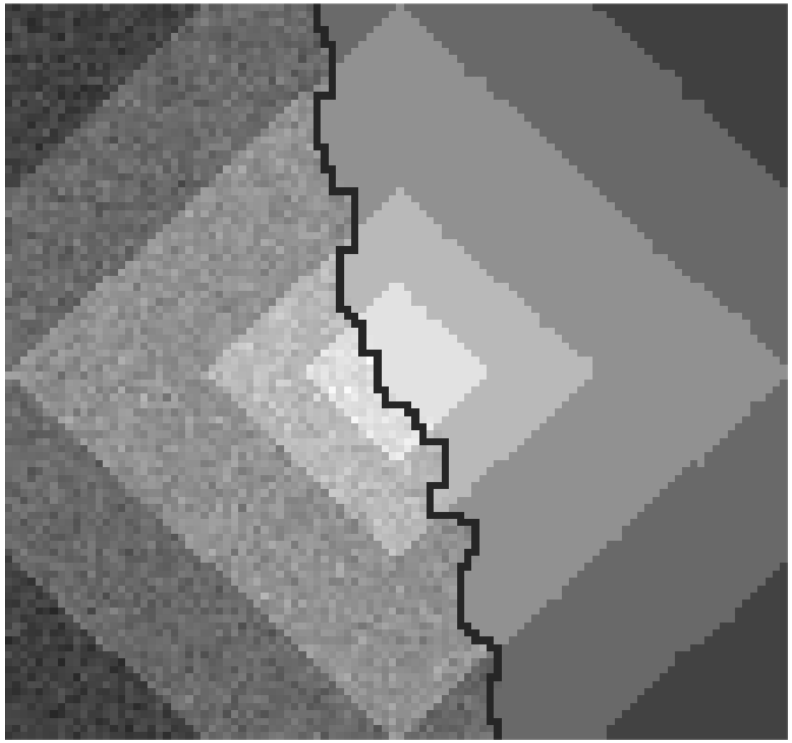


Figure 16: “Restoration” of noisy image.

Stereo disparity

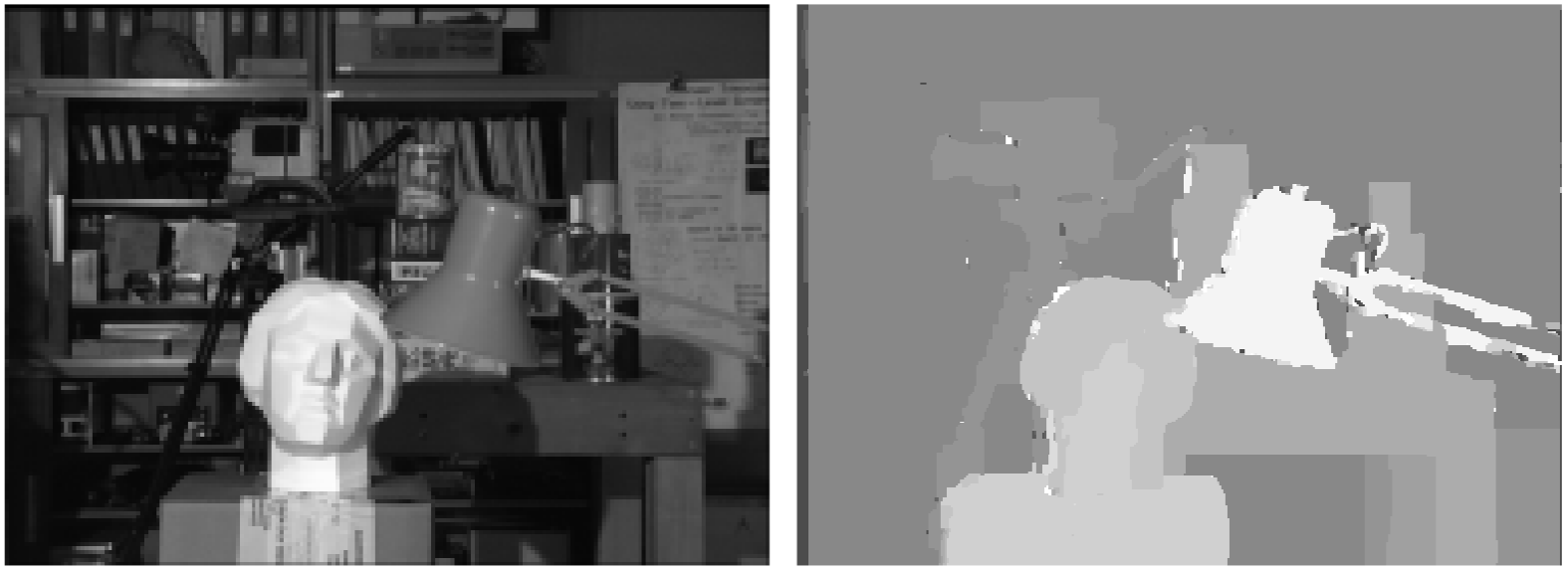


Figure 17: Left: One image in a stereo pair. Right: Depth estimated with graph cuts.

More than two terminals?

- The examples on the previous slide have more than two types of labels.
- So far, we have only considered minimal graph cuts on graphs with two terminals. Thus, we can only compute binary labelings.
- Unfortunately, computing globally minimal graph cuts for more than two terminals is NP-hard.
- In the next lecture, we will look at a strategy for computing approximate solutions to this problem.

Comparison, min-cuts and MSF-cuts

MSF-cuts

- Advantages
  - Globally optimal, according to the max-norm.
  - Fast computiation.
  - Seed-relative robustness.
  - Handles any number of labels.
- Drawbacks
  - No “smoothness term”, sensitive to noise.

Comparison, min-cuts and MSF-cuts

Min-cuts

- Advantages
  - Globally optimal, according to the 1-norm.
  - Can be used to approximate continuous cut metrics.
  - Can be computed in polynomial time.
- Drawbacks
  - Restricted to binary labeling.
  - Slower to compute than MSFs.

References

[1] Y. Boykov and V. Kolmogorov.

An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision.

IEEE PAMI, 26(9):1124–1137, 2004.

[2] Yuri Boykov.

Computing geodesics and minimal surfaces via graph cuts.

In *International Conference on Computer Vision*, pages 26–33, 2003.

[3] J. Edmonds and R. Karp.

Theoretical improvements in algorithmic efficiency for network flow problems.

Journal of the ACM, 19(2), 1972.

[4] A.P. Eriksson, C. Olsson, and F. Kahl.

Normalized cuts revisited: A reformulation for segmentation with linear grouping constraints.

Journal of Mathematical Imaging and Vision, 39(1):45–61, 2011.

[5] L. Ford and D. Fulkerson.

Flows in networks.

Princeton University Press, 1962.

[6] Jianbo Shi and J. Malik.

Normalized cuts and image segmentation.

Pattern Analysis and Machine Intelligence, IEEE Transactions on, 22(8):888 –905, 2000.