Maximum flow - Minimum cut (Part II)

Quan Hoang

November 22-29, 2018

Outline

Extensions of max flow problem

Appplications of max flow - min cut problem

Real coding

Extensions of max flow problem

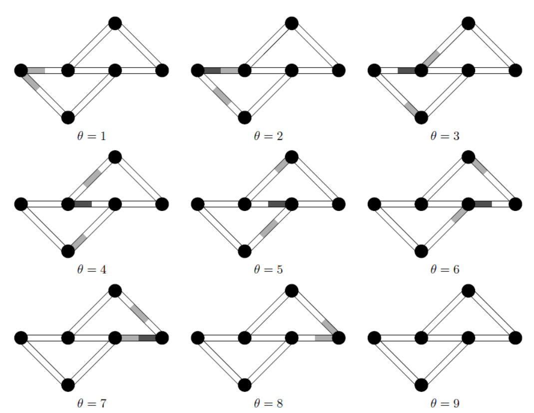
Some solved cases

- ⋆ Vertex with capacity.
- ⋆ Maximum flow in undirected graph.
- ⋆ Multi-sources and multi-sinks[1]

[1] explained in the Application section

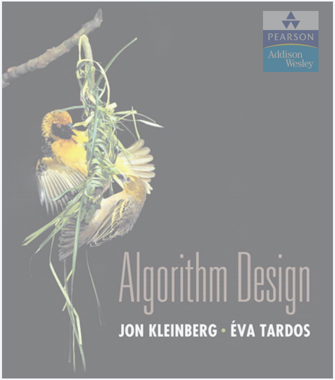
Other cases

⋆ **Time related flow.**



⋆ **Sink nodes with desired source nodes.**

Appplications of max flow - min cut problem



SECTION 7.5

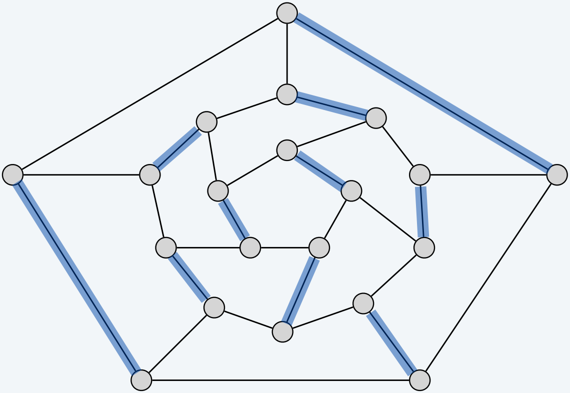
7. NETWORK FLOW II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Matching

Def. Given an undirected graph $G = (V, E)$, subset of edges $M \subseteq E$ is a matching if each node appears in at most one edge in M.

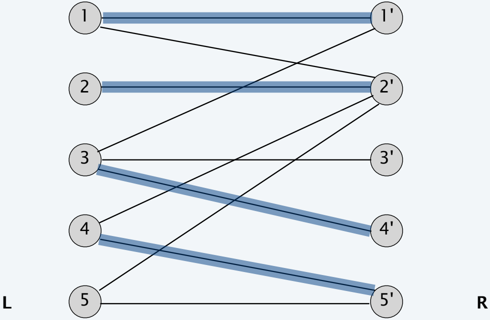
Max matching. Given a graph G, find a max-cardinality matching.



Bipartite matching

Def. A graph G is bipartite if the nodes can be partitioned into two subsets L and R such that every edge connects a node in L with a node in R.

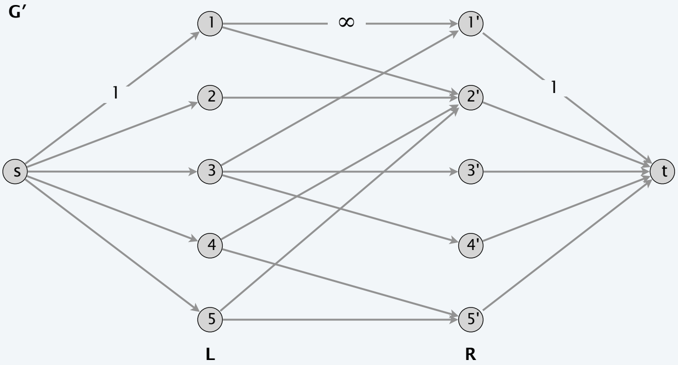
Bipartite matching. Given a bipartite graph $G = (L \cup R, E)$, find a max-cardinality matching.



matching: 1–1', 2–2', 3–4', 4–5'

Bipartite matching: max-flow formulation

- Create digraph $G' = (L \cup R \cup \{s, t\},\ E')$.
- Direct all edges from L to R, and assign infinite (or unit) capacity.
- Add unit-capacity edges from s to each node in L.
- Add unit-capacity edges from each node in R to t.



Nonbipartite matching

Problem. Given an undirected graph, find a max-cardinality matching.

- Structure of nonbipartite graphs is more complicated.
- But well understood. [Tutte–Berge formula, Edmonds–Galai]
- Blossom algorithm: $O(n^4)$. [Edmonds 1965]
- Best known: $O(m\, n^{1/2})$. [Micali–Vazirani 1980, Vazirani 1994]

PATHS, TREES, AND FLOWERS

JACK EDMONDS

1. Introduction. A *graph* G for purposes here is a finite set of elements called *vertices* and a finite set of elements called *edges* such that each edge *meets* exactly two vertices, called the *end-points* of the edge. An edge is said to *join* its end-points.

A *matching* in G is a subset of its edges such that no two meet the same vertex. We describe an efficient algorithm for finding in a given graph a matching of maximum cardinality. This problem was posed and partly solved by C. Berge; see Sections 3.7 and 3.8.

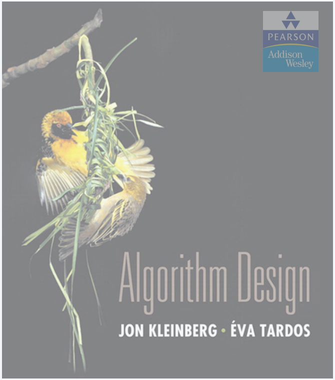
COMBINATORICA
Akadémiai Kiadó – Springer-Verlag

COMBINATORICA 14 (1) (1994) 71–109

A THEORY OF ALTERNATING PATHS AND BLOSSOMS FOR PROVING CORRECTNESS OF THE $O(\sqrt{V}E)$ GENERAL GRAPH MAXIMUM MATCHING ALGORITHM

VIJAY V. VAZIRANI[1]

Received December 30, 1989
Revised June 15, 1993



SECTION 7.6

7. NETWORK FLOW II

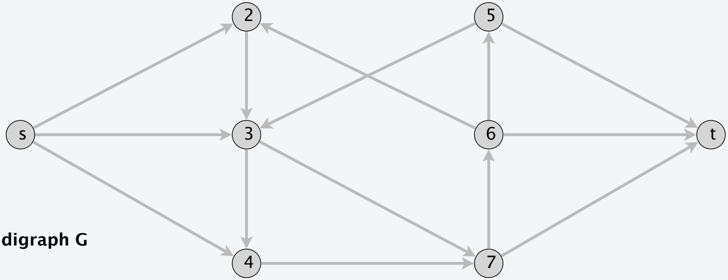
- *bipartite matching*
- ***disjoint paths***
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Edge-disjoint paths

Def. Two paths are edge-disjoint if they have no edge in common.

Edge-disjoint paths problem. Given a digraph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint $s \rightarrow t$ paths.

Ex. Communication networks.



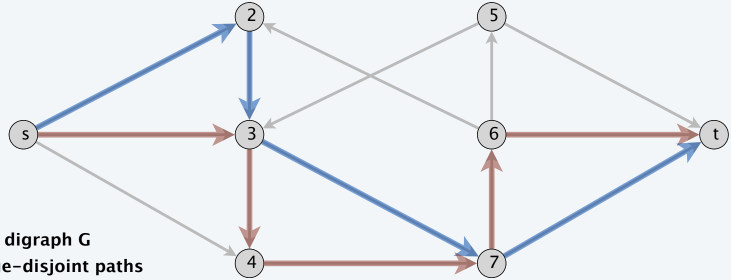
digraph G

Edge-disjoint paths

Def. Two paths are edge-disjoint if they have no edge in common.

Edge-disjoint paths problem. Given a digraph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint $s \rightarrow t$ paths.

Ex. Communication networks.



digraph G
2 edge-disjoint paths

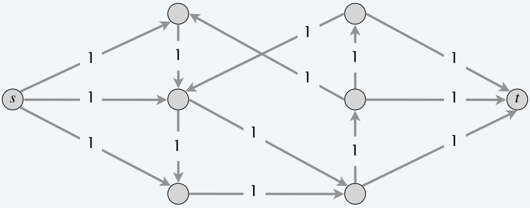
Edge-disjoint paths

Max-flow formulation. Assign unit capacity to every edge.

Theorem. Max number of edge-disjoint $s \rightsquigarrow t$ paths = value of max flow.

Pf. $\leq$

- Suppose there are k edge-disjoint $s \rightsquigarrow t$ paths $P_1, \ldots, P_k$.
- Set $f(e) = 1$ if e participates in some path P_j ; else set $f(e) = 0$.
- Since paths are edge-disjoint, f is a flow of value k. ▪



Edge-disjoint paths

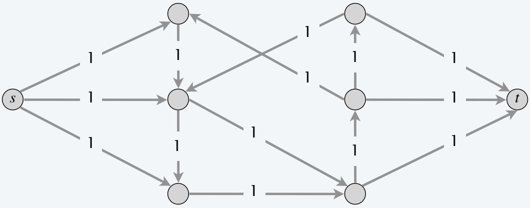
Max-flow formulation. Assign unit capacity to every edge.

Theorem. Max number of edge-disjoint $s \to t$ paths = value of max flow.

Pf. $\geq$

- Suppose max flow value is k.
- Integrality theorem $\Rightarrow$ there exists 0–1 flow f of value k.
- Consider edge (s, u) with $f(s, u) = 1$.
  - by flow conservation, there exists an edge (u, v) with $f(u, v) = 1$
  - continue until reach t, always choosing a new edge
- Produces k (not necessarily simple) edge-disjoint paths. ▪

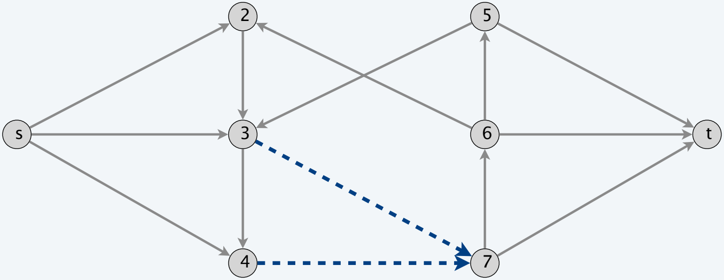
can eliminate cycles to get simple paths in $O(mn)$ time if desired (flow decomposition)



Network connectivity

Def. A set of edges $F \subseteq E$ disconnects t from s if every $s \rightarrow t$ path uses at least one edge in F.

Network connectivity. Given a digraph $G = (V, E)$ and two nodes s and t, find min number of edges whose removal disconnects t from s.

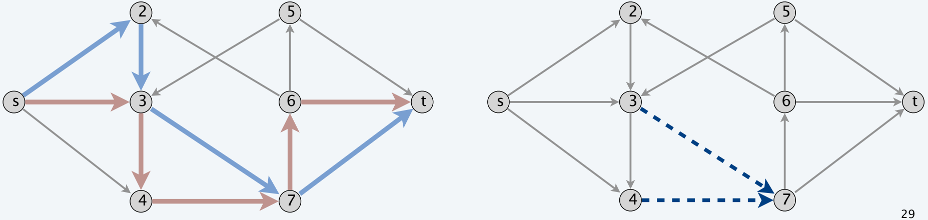


Menger's theorem

Theorem. [Menger 1927] The max number of edge-disjoint $s \to t$ paths equals the min number of edges whose removal disconnects t from s.

Pf. $\leq$

- Suppose the removal of $F \subseteq E$ disconnects t from s, and $|F| = k$.
- Every $s \to t$ path uses at least one edge in F.
- Hence, the number of edge-disjoint paths is $\leq k$. ▪

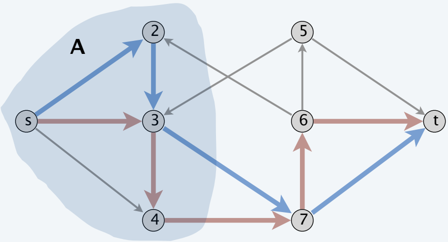


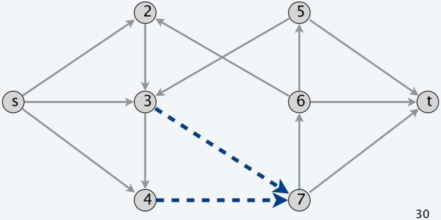
Menger's theorem

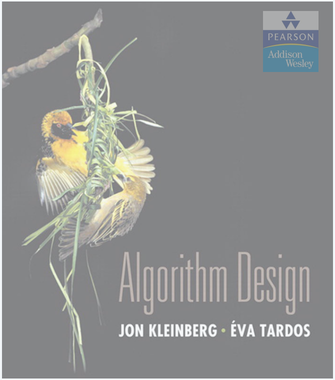
Theorem. [Menger 1927] The max number of edge-disjoint $s \to t$ paths equals the min number of edges whose removal disconnects t from s.

Pf. $\geq$

- Suppose max number of edge-disjoint paths is k.
- Then value of max flow $= k$.
- Max-flow min-cut theorem $\Rightarrow$ there exists a cut (A, B) of capacity k.
- Let F be set of edges going from A to B.
- $|F| = k$ and disconnects t from s. ▪







SECTION 7.7

7. NETWORK FLOW II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Which extensions to max flow can be easily modeled?

A. Multiple sources and multiple sinks.

B. Undirected graphs.

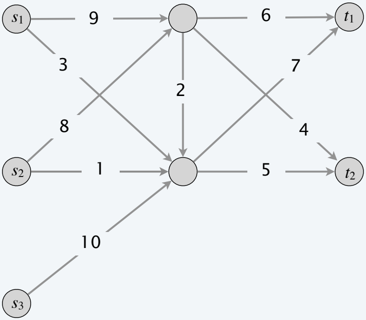
C. Lower bounds on edge flows.

D. All of the above.

Multiple sources and sinks

Def. Given a digraph $G = (V, E)$ with edge capacities $c(e) \geq 0$ and multiple source nodes and multiple sink nodes, find max flow that can be sent from the source nodes to the sink nodes.

flow network G

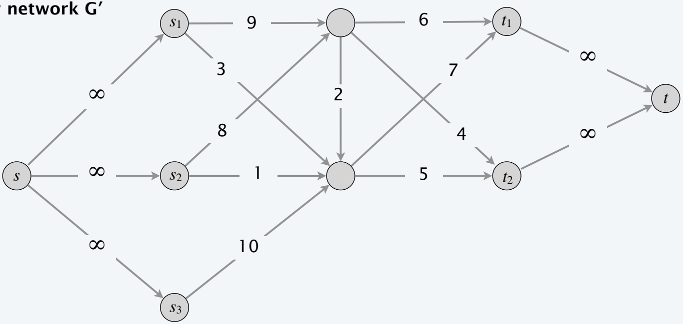


Multiple sources and sinks: max-flow formulation

- Add a new source node s and sink node t.
- For each original source node s_i add edge (s, s_i) with capacity ∞.
- For each original sink node t_j, add edge (t_j, t) with capacity ∞.

Claim. 1–1 correspondence betweens flows in G and G'.

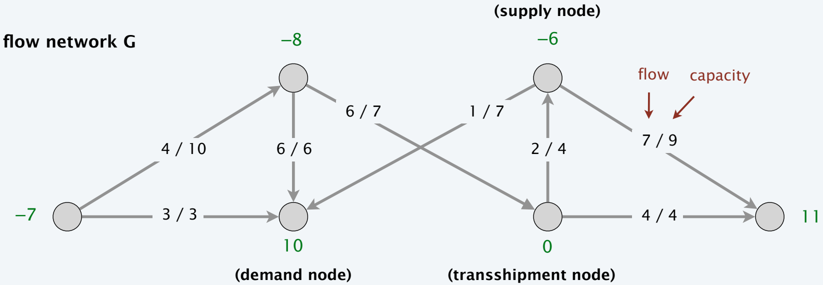
flow network G′



Circulation with supplies and demands

Def. Given a digraph $G = (V, E)$ with edge capacities $c(e) \geq 0$ and node demands $d(v)$, a circulation is a function $f(e)$ that satisfies:

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (flow conservation)



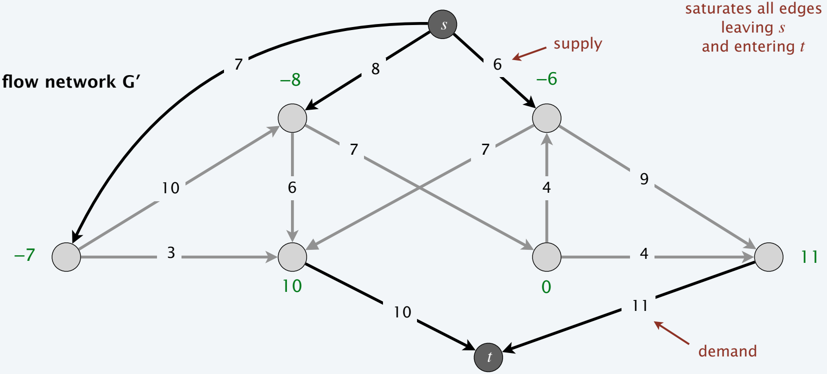
Circulation with supplies and demands: max-flow formulation

- Add new source s and sink t.
- For each v with $d(v) < 0$, add edge (s, v) with capacity $-d(v)$.
- For each v with $d(v) > 0$, add edge (v, t) with capacity $d(v)$.

Claim. G has circulation iff G' has max flow of value $D = \sum_{v\,:\,d(v)>0} d(v) = \sum_{v\,:\,d(v)<0} -d(v)$

saturates all edges leaving s and entering t

flow network G′



Circulation with supplies and demands

Integrality theorem. If all capacities and demands are integers, and there exists a circulation, then there exists one that is integer-valued.

Pf. Follows from max-flow formulation + integrality theorem for max flow.

Theorem. Given (V, E, c, d), there does not exist a circulation iff there exists a node partition (A, B) such that $\Sigma_{v \in B} d(v) > cap(A, B)$.

demand by nodes in B exceeds supply of nodes in B plus max capacity of edges going from A to B

Pf sketch. Look at min cut in G'.

Circulation with supplies, demands, and lower bounds

Def. Given a digraph $G = (V, E)$ with edge capacities $c(e) \geq 0$, lower bounds $\ell(e) \geq 0$, and node demands $d(v)$, a circulation $f(e)$ is a function that satisfies:

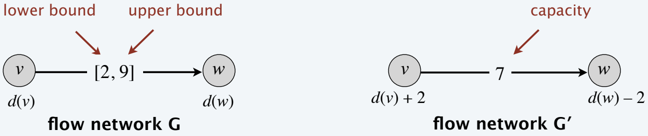
- For each $e \in E$: $\ell(e) \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (flow conservation)

Circulation problem with lower bounds. Given (V, E, ℓ, c, d), does there exist a feasible circulation?

Circulation with supplies, demands, and lower bounds

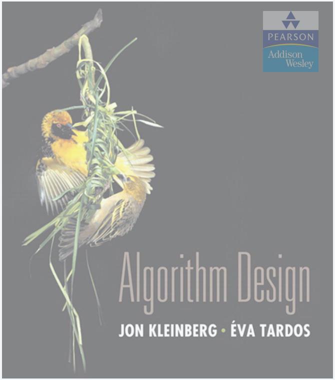
Max-flow formulation. Model lower bounds as circulation with demands.

- Send $\ell(e)$ units of flow along edge e.
- Update demands of both endpoints.



Theorem. There exists a circulation in G iff there exists a circulation in G'. Moreover, if all demands, capacities, and lower bounds in G are integers, then there exists a circulation in G that is integer-valued.

Pf sketch. $f(e)$ is a circulation in G iff $f'(e) = f(e) - \ell(e)$ is a circulation in G'.



SECTION 7.10

7. NETWORK FLOW II

- bipartite matching
- disjoint paths
- extensions to max flow
- survey design
- airline scheduling
- *image segmentation*
- project selection
- baseball elimination

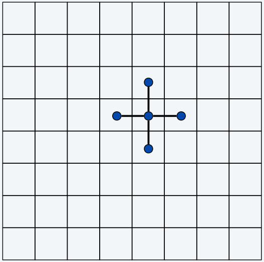
Image segmentation

Image segmentation.

- Divide image into coherent regions.
- Central problem in image processing.

Ex. Separate human and robot from background scene.

Image segmentation



Foreground / background segmentation.

- Label each pixel in picture as belonging to foreground or background.
- V = set of pixels, E = pairs of neighboring pixels.
- $a_i \ge 0$ is likelihood pixel i in foreground.
- $b_i \ge 0$ is likelihood pixel i in background.
- $p_{ij} \ge 0$ is separation penalty for labeling one of i and j as foreground, and the other as background.

Goals.

- Accuracy: if $a_i > b_i$ in isolation, prefer to label i in foreground.
- Smoothness: if many neighbors of i are labeled foreground, we should be inclined to label i as foreground.
- Find partition (A, B) that maximizes: $$\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

  ↗ foreground ↖ background

Image segmentation

Formulate as min-cut problem.

- Maximization.
- No source or sink.
- Undirected graph.

Turn into minimization problem.

- Maximizing $$\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

- is equivalent to minimizing

$$\left(\sum_{i \in V} a_i + \sum_{j \in V} b_j \right) - \sum_{i \in A} a_i - \sum_{j \in B} b_j + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

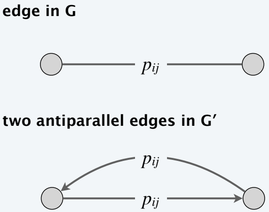
a constant

- or alternatively $$\sum_{j \in B} a_j + \sum_{i \in A} b_i + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

Image segmentation

Formulate as min-cut problem $G' = (V', E')$.

- Include node for each pixel.
- Use two antiparallel edges instead of undirected edge.
- Add source s to correspond to foreground.
- Add sink t to correspond to background.



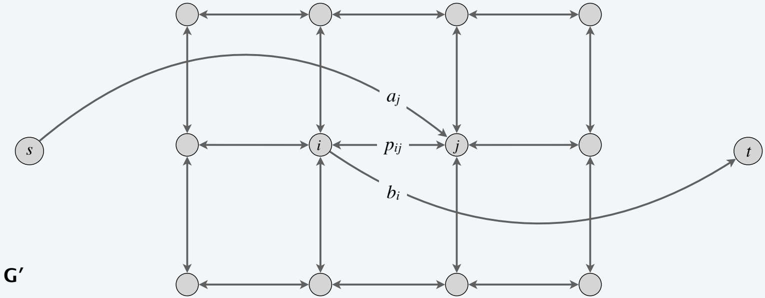


Image segmentation

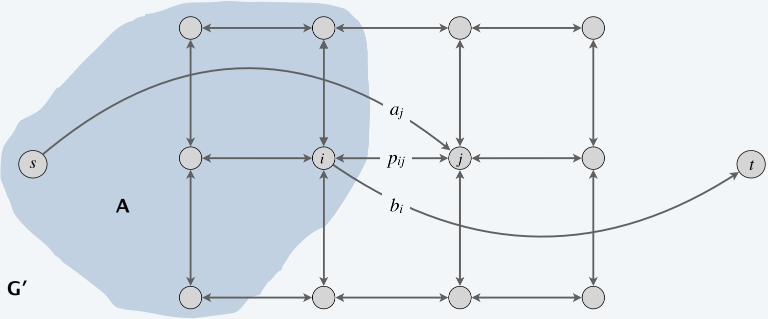
Consider min cut (A, B) in G'.

- A = foreground.

$$cap(A,B) \;=\; \sum_{j \in B} a_j \;+\; \sum_{i \in A} b_i \;+\; \sum_{\substack{(i,j) \in E \\ i \in A,\; j \in B}} p_{ij}$$

← if i and j on different sides, p_{ij} counted exactly once

- Precisely the quantity we want to minimize.



Grabcut image segmentation

Grabcut. [Rother–Kolmogorov–Blake 2004]

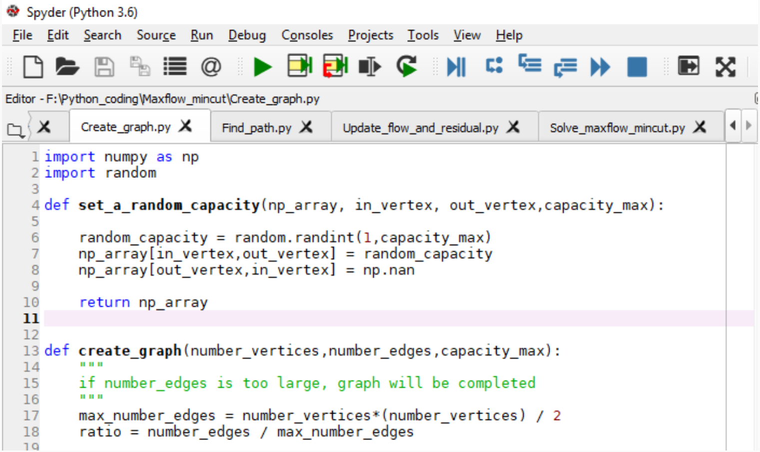
"GrabCut" — Interactive Foreground Extraction using Iterated Graph Cuts

Carsten Rother* Vladimir Kolmogorov† Andrew Blake‡
Microsoft Research Cambridge, UK

Figure 1: **Three examples of GrabCut .** The user drags a rectangle loosely around an object. The object is then extracted automatically.

Real coding

Spyder



Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - F:\Python_coding\Maxflow_mincut\Create_graph.py
Create_graph.py
Find_path.py
Update_flow_and_residual.py
Solve_maxflow_mincut.py
import numpy as np
import random

def set_a_random_capacity(np_array, in_vertex, out_vertex,capacity_max):

    random_capacity = random.randint(1,capacity_max)
    np_array[in_vertex,out_vertex] = random_capacity
    np_array[out_vertex,in_vertex] = np.nan

    return np_array

def create_graph(number_vertices,number_edges,capacity_max):
    """
    if number_edges is too large, graph will be completed
    """
    max_number_edges = number_vertices*(number_vertices) / 2
    ratio = number_edges / max_number_edges

References

Robert Sedgewick & Kevin Wayne. Algorithms, 4th Edition. Addison-Wesley.
Slides at https://algs4.cs.princeton.edu/lectures/

Kleinberg, J., & Tardos, E. (2006). The algorithm design. Pearson, Addison-Wesley.
Slides at https://www.cs.princeton.edu/∼wayne/kleinberg-tardos/

Martin Skutella. An Introduction to Network Flows over time.