Graphs, graph algorithms
(e.g. for image segmentation) ... in progress

Václav Hlaváč

Czech Technical University in Prague
Czech Institute of Informatics, Robotics and Cybernetics
166 36 Prague 6, Jugoslávských partyzánů 1580/3, Czech Republic
http://people.ciirc.cvut.cz/hlavac, vaclav.hlavac@cvut.cz
also Center for Machine Perception, http://cmp.felk.cvut.cz

Courtesy: Jaehyun Park, Jianbo Shi

Outline of the talk:

◆ Graph-based image segmentation, ideas
◆ Graphs, concepts
◆ Flow network
◆ Shortest path (Dijkstra algorithm)
◆ Minimum spanning tree
Graph-based image segmentation, main ideas

- Convert an image into a graph
  - Graph vertices correspond to individual pixels.
  - Edges connect neighboring pixels.
  - Additional graph vertices and edges encode other constraints. 
    *Example:* a special node (source) denotes objects and a special node (sink) denotes background in object/background segmentation.
    *The source/sink concepts come from flow networks.*

- Manipulate the graph to segment the image.
'Related seminal papers'

  
  - $A$: Pixel classified as object or background. Novelty: adding interactivity.
  
  - Minimize energy function $E(A) = B(A) + \lambda R(A)$, where $B(A) =$ the cost of all edges between object pixels and background pixels; $R(A) =$ the cost of deciding if a pixel is object or background.

  
  - Cluster the vertices based on edge weight.

- C. Rother, V. Kolmogorov, A. Blake: GrabCut: Interactive Foreground Extraction using Iterated Graph Cuts. ACM Transactions on Graphics (SIGGRAPH’04), 2004.
Prerequisites

- Time or memory asymptotic computational complexity:
  \( \mathcal{O}(\cdot) \) bounds from below; i.e. not more than \ldots in the worst case;
  \( \Omega(\cdot) \) bounds from above; i.e. not less than \ldots in the worst case.
An image represented as a graph

- Nodes correspond to pixels.
- Edges connect neighboring pixels. 4-neighbors are considered in the example.
- Edges weights express the similarity between the neighboring pixels (a binary relation).
Image regions, region adjacency graph

- Pixels are grouped to larger regions in the process of (iterative) segmentation.

- Superpixels are regions grouped from pixels of similar intensity or color, in which interpretation is not taken into account.

- Region Adjacency Graph.

![input image](image1.png) ![regions (here superpixels)](image2.png) ![RAG](image3.png)

Courtesy: Images by Vighnesh Birodkar.
Graphs, concepts

- It is assumed that a student has studied related graph theory elsewhere. Main concepts are reminded here only.

- [http://web.stanford.edu/class/cs97si/](http://web.stanford.edu/class/cs97si/), lecture 6, 7
- [https://www.cs.indiana.edu/~achauhan/Teaching/B403/LectureNotes/10-graphalgo.html](https://www.cs.indiana.edu/~achauhan/Teaching/B403/LectureNotes/10-graphalgo.html)

Concepts
- Graphs: directed, undirected
- Adjacency, similarity matrices
- Degree, volume of a vertex
- Graph cut

Related graph algorithms
- Minimum spanning tree
- Shortest path
- Max graph flow = min graph cut
Undirected/directed graph

Undirected graph

- $G = (V, E)$ is composed of vertices $V$ and undirected edges $E \subseteq V \times V$ representing an unordered relation between two vertices.
- No self-loops.
- The number of edges $|E| = \mathcal{O}(|V|^2)$.

Directed graph

- $G = (V, E)$ is composed of vertices $V$ and directed edges $E$ representing an ordered relation between two vertices.
- Oriented edge $e = (u, v)$ has the tail $u$ and the head $v$ (shown as the arrow $\rightarrow$). The edge $e$ is different from $e' = (v, u)$ in general.
A *weighted graph* associates weights with either the edges or the vertices or both. *E.g., a road map: graph edges are weighted with distances.*
Degree of a vertex

- The degree of vertex \( i \) is the number of graph edges incident on vertex \( i \).
- Directed graphs have in-degree, out-degree.

Degree of node:  
\[ d_i = \sum_j S_{ij} \]  

Courtesy: Jinbo Shi
Volume of a set

$$\text{vol}(A) = \sum_{i \in A} d_i, A \subseteq V$$

Courtesy: Jinbo Shi
Dense/sparse graph

- Running times are typically expressed in terms of $|E|$ and $|V|$ (often dropping the “’s”).
- The graph is dense if $|E| \approx |V|^2$.
- The graph is sparse if $|E| \approx |V|$.
- Many interesting graphs are sparse. 
  *E.g., planar graphs, in which no edges cross, have $|E| = \mathcal{O}(|V|)$ by Euler’s formula.*
- If you know you are dealing with dense or sparse graphs, different data structures may make sense.
Graph represented by the adjacency matrix

- **Adjacency relationship** is:
  - Symmetric if the graph $G$ is undirected.
  - Not necessarily so if $G$ is directed.

- **Adjacency matrix** $A$ has elements $a_{ij}$, $i, j = 1, \ldots, |V|$.
  - Elements $a_{ij} = 1$ if graph vertices $i, j$ share an edge; 0 otherwise.
  - Uses $\Omega(|V|)$ memory.

\[
\begin{array}{cccc}
  a & b & c & d \\
  a & 0 & 1 & 1 & 0 \\
b & 1 & 0 & 1 & 0 \\
c & 1 & 1 & 0 & 1 \\
d & 0 & 0 & 1 & 0 \\
\end{array}
\]
In the case of the weighted graph, the adjacency matrix is generalized to the similarity matrix $S$ of the graph.

Elements of $A$ have value of the weight of the edge between two respective vertices, $a_{ij} = w_{ij}$.

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>
Similarity matrix $S = [S_{ij}]$

is generalized adjacency matrix
Graph represented by the adjacency list

- Adjacency list: For each vertex \( v \in V \), store a list of vertices adjacent to \( v \) (in other words: going out of \( v \)).
  - Easy to iterate over edges incident to a certain node.
  - The lists have variable lengths.
  - Asymptotic bound of the space usage from above: \( \Omega(|E| + |V|) \).

- Our example:

  ![Graph diagram]

  - \( \text{Adj}[a] = \{a, c\} \)
  - \( \text{Adj}[b] = \{c\} \)
  - \( \text{Adj}[c] = \{\} \)
  - \( \text{Adj}[d] = \{c\} \)

- Variation: For oriented graphs, it is possible to store the list of graph edges coming into the vertex.
Implementing adjacency list

Three solutions:

1. Using linked lists
   - Too much memory / time overhead.
   - Using dynamically allocated memory or pointers is bad.

2. Using arrays of vectors
   - Easier to code, no bad memory issues.
   - Very slow.

3. Using arrays
   - Assuming the total number of edges $|E|$ is known.
   - Very fast and memory efficient.
Implementation using arrays, example

![Graph Image]

<table>
<thead>
<tr>
<th>ID</th>
<th>To</th>
<th>Next Edge ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>From</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last Edge ID</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>-</td>
</tr>
</tbody>
</table>

Courtesy: Jaehyun Park, Stanford University
Graphs. Implementation using arrays (1)

- Have two arrays; E of size $|E|$ and LE of size $|V|$.
  - E contains the edges.
  - LE contains the starting pointers of the edge lists.
- Initialize LE[i] = -1 for all i
  - LE[i] = 0 is also fine if the arrays are 1-indexed.
- Inserting a new edge from vertex u to vertex v with ID k
  - E[k].to = v
  - E[k].nextID = LE[u]
  - LE[u] = k
- Iterating over all edges starting at u
  - for (ID = LE[u]; ID != -1; ID = E[ID].nextID
    // E[ID] is an edge starting from u
Once built, it is hard to modify the edges.

The graph better be static!

However, adding more edges is easy.
Special graphs. Multigraph, bipartite graph

- A **multigraph** allows multiple edges between the same vertices. *E.g., the call graph in a program (a function can get called from multiple points in another function).*

- A **bipartite graph** has graph vertices decomposed into two disjoint sets such that no two graph vertices within the same set are adjacent.

- Bipartite graphs are equivalent to two-colorable graphs.

- A bipartite graph is a special case of a $k$-partite graph with $k = 2$. 

Courtesy: Wolfram MathWorld.
Path, graph cycle

- A path from a vertex $u$ to a vertex $v$ is a sequence $(v_0, v_1, \ldots, v_k)$ of vertices where $v_0 = v$, $v_k = u$, and $(v_i, v_{i+1}) \in E$ for $i = 0, 1, \ldots, k - 1$.

- A cycle of a graph $G$ is a subset of the edge set of $G$ that forms a path such that the first node of the path corresponds to the last node.

- Hamiltonian cycle is a cycle that uses each graph vertex of a graph exactly once. A very hard problem. Still unsolved.

- Eulerian cycle (for undirected connected graphs) is a sequence of vertices that visits every edge exactly once and comes back to the starting vertex.
  
  - A single stroke/line drawing task.
  
  - Precondition: Each vertex has an even degree. L. Euler formulated this precondition on 7 bridges of Königsberg example in ??.
A simple path is a path, in which all vertices, except possibly the first and the last, are different.

The length of a path is defined as the number of edges in the path.

If the graph is weighted then the length of the path is the sum of edges weights in the path.

If the graph $G$ is connected then there is a path between every pair of vertices.

$|E| \geq |V| - 1$. 
Dijkstra algorithm for the shortest path in general graphs

- $s$ – source node
- $d(j)$ – the minimal distance from node $s$ to node $j$
- $\text{pred}(j)$ – predecessor of the node $j$

**Dijkstra algorithm** (1956) finds the shortest node from the source node $s$ to all nodes.

% $N$ – set of all graph nodes

$V := s$ % visited nodes;

$U := N \setminus s$ % unvisited nodes;

$d(s) := 0$, $i := s$;

**while** $|V| < |N|$ **do**

choose $(i, j)$:

\[
d(j) := \min_{k, m} \{d(k) + c_{km} \mid k \in V, m \in U\};
\]

$U = U \setminus \{j\}$;

$V = V + \{j\}$;

$\text{pred}(j) := i$;

**end**

Performance $O(|N|^2)$ with the heap reshuffling $O(|N| \log |N|)$. 
A special graph: acyclic graph

- A graph containing no cycles of any length is known as an **acyclic graph**. Other graphs are cyclic.

- An acyclic graph is bipartite.

- A cyclic graph is bipartite iff all its cycles are of even length.

Special graphs: forest, tree

- A **forest** is an acyclic graph.
- A **tree** is a connected acyclic graph.
- Tree is the most important type of special graphs because many problems are easy to solve on trees.

Alternative equivalent tree definitions:

- A connected graph $G$ with $|E| = |V| − 1$ edges.
- An acyclic graph with $|V| − 1$ edges.
- There is exactly one path between every pair of vertices.
- An acyclic graph but adding any edge results in a cycle.
- A connected graph but removing any edge disconnects it.
Cut in a graph

\[ \text{cut}(A, \bar{A}) = \sum_{i \in A, j \in \bar{A}} S_{i,j} \]

Courtesy: Jinbo Shi
A cut is a set of edges $C \subset E$ such that two vertices (called terminals) became separated on the induced graph $G' = (V, E \setminus C)$.

Denoting a source terminal as $s$ and a sink terminal as $t$, a cut $(S, T)$ of $G = (V, E)$ is a partition of $V$ into $S$ and $T = V \setminus S$, such that $s \in S$ and $t \in T$. 
Visual summary of a graph terminology

Similarity matrix $S = [ S_{ij} ]$

Degree of node: $d_i = \sum_j S_{ij}$

Volume of set:

Graph Cuts

Courtesy: Jinbo Shi
A flow network is a directed graph with nonnegative edge weights (called also capacities).

A flow $f$ is a real-valued (often integer) function, which satisfies the following three properties:

1. Capacity $c$ constraint
   For all $u, v \in V$, $f(u, v) \leq c(u, v)$.

2. Skew symmetry
   For all $u, v \in V$, $f(u, v) = -f(v, u)$.

3. Flow conservation
   For all $u \in (V \setminus \{s, t\})$, $\sum_{v \in V} f(u, v) = 0$. 
**Spanning tree**

- **Definition:** A spanning tree $T$ of an undirected graph $G$ is a subgraph that is a tree which includes all of the vertices of $G$.

- In general, a graph may have several spanning trees.

- If a graph that is not connected will not contain a spanning tree. (cf. spanning forest).

- If all of the edges of $G$ are also edges of a spanning tree $T$ of $G$, then $G$ is a tree and is identical to $T$ (that is, a tree has a unique spanning tree and it is itself).

*One of several possible spanning trees of a 4 $\times$ 4 grid with 4-neighborhood.*

Graph coloring, task formulation

ToDo.
Minimum Spanning Tree

- **Given**: Given an undirected weighted graph $G = (V, E)$.
- **Task**: Find a subset of $E$ with the minimum total weight that connects all the nodes into a tree.

---

**Kruskal's algorithm**

- Takes $\mathcal{O}(|E| \log |E|)$ time.
- Easy to code.
- Generally slower than Prim’s algorithm.

**Prim’s algorithm**

- Time complexity depends on the implementation. Can be $\mathcal{O}(|V|^2 + |E|)$, $\mathcal{O}(|E| \log |V|)$, or $\mathcal{O}(|E| + |V| \log |V|)$.
- More difficult to code.