Graph Based Image Processing and Combinatorial Optimization

2021

Filip Malmberg

About the course

- Course webpage: `http://www.cb.uu.se/~filip/GraphBasedImageProcessing2021/`
- Format:
  - Lectures.
  - Reading assignments before each lecture (except this first one)
- No mandatory attendance.
- Examination: A set of tasks to be completed (programming tasks, answering theoretical questions)

Course goals

After completing the course, you should:

- be familiar with basic graph theory and how it applies to image processing.
- have a good understanding of combinatorial optimization.
- have a good understanding of state-of-the-art methods for solving combinatorial optimization problems arising in image processing.
- have some experience implementing and using such methods, and applying them in your own research.

Lecture and reading assignments

- For every lecture, I will put some papers on the webpage for your to read before the lecture.
- This is not mandatory, but please do it! We will have more interesting discussions during the lectures, and you will benefit more from the course.

Examination: Tasks

To recieve credits for the course, you should:

- Complete a series of tasks, both practical (programming) and theoretical.
- The instructions for the tasks will be published on the webpage. The tasks can be completed at any time during the course, but there will be notes in the instructions on what lectures each task depends on.
- You are allowed to work in groups in groups (up to 3 people) on the assignments. Some programming experience is required, try to form teams that cover this!
- For the programming parts, I will give you some C++ code to start from, but feel free to use another language that you are more comfortable with!
- Submit your solutions (code+written report) to me (Filip).

Background: Graph based image processing

- Graphs provide a powerful unified representation for image analysis and processing. In this course, we will give an overview of some of the most important results, and the state-of-the-art in this field.
- How and why do we represent images as graphs?
- Graph-based methods for:
    - Image segmentation/Classification/Clustering
    - Image restoration/filtering
    - Image registration / stereo matching
    - Scattered data interpolation

What is an image?

"We will sometimes regard a *picture* as being a real-valued, non-negative function of two real variables; the value of this function at a point will be called the *gray-level* of the picture at the point."

Rosenfeld, *Picture Processing by Computer*, ACM Computing Surveys, 1969.

What is a digital image?

Storing the (continuous) image in a computer requires digitization, e.g.

- Sampling (recording image values at a finite set of *sampling points*).
- Quantization (discretizing the continuous function values).

Typically, sampling points are located on a Cartesian grid.

Generalized images

This basic model can be generalized in several ways:

- Generalized image modalities (e.g., multispectral images)
- Generalized image domains (e.g. video, volume images)
- Generalized sampling point distributions (e.g. non-Cartesian grids)

The methods we develop in image analysis should (ideally) be able to handle this.

Why graph-based?

- Discrete and mathematically simple representation that lends itself well to the development of efficient and provably correct methods.
- A minimalistic image representation – flexibility in representing different types of images.
- A *lot* of work has been done on graph theory in other applications, We can re-use existing algorithms and theorems developed for other fields in image analysis!

Graphs, basic definition

- A graph is a pair $G = (V, E)$, where
  - V is a set.
  - E consists of pairs of elements in V.
- The elements of V are called the *vertices* of G.
- The elements of E are called the *edges* of G.

Graphs basic definition

- An edge spanning two vertices v and w is denoted $e_{v,w}$.
- If $e_{v,w} \in E$, we say that v and w are *adjacent*.
- The set of vertices adjacent to v is denoted $\mathcal{N}(v)$.

Example

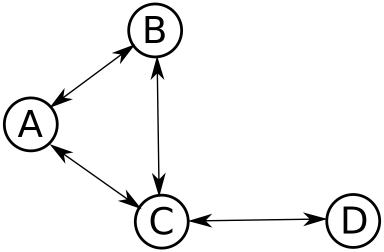

Figure 1: A drawing of an undirected graph with four vertices $\{A, B, C, D\}$ and four edges $\{e_{A,B},\ e_{A,C},\ e_{B,C},\ e_{C,D}\}$.

Example

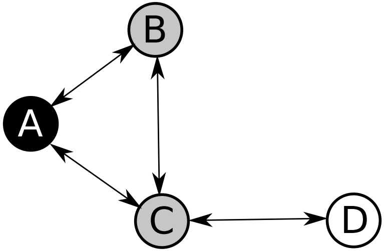

Figure 2: The set $\mathcal{N}(A) = \{B, C\}$ of vertices adjacent to A.

Images as graphs

- Graph based image processing methods typically operate on *pixel adjacency graphs*, i.e., graphs whose vertex set is the set of image elements, and whose edge set is given by an adjacency relation on the image elements.
- Commonly, the edge set is defined as all vertices v, w such that

$$d(v, w) \leq \rho \; . \tag{1}$$

- This is called the *Euclidean adjacency relation*.

Pixel adjacency graphs, 2D

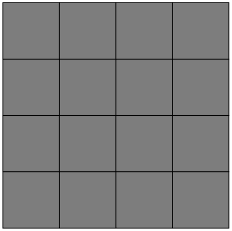
Figure 3: A 2D image with 4×4 pixels.

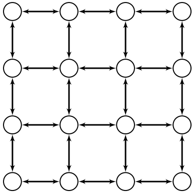
Figure 4: A 4-connected pixel adjacency graph.

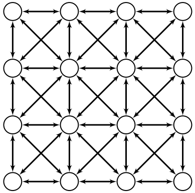
Figure 5: A 8-connected pixel adjacency graph.

Pixel adjacency graphs, 3D

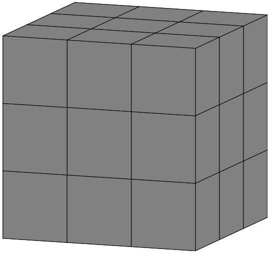
Figure 6: A volume image with $3 \times 3 \times 3$ voxels.

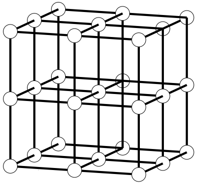
Figure 7: A 6-connected voxel adjacency graph.

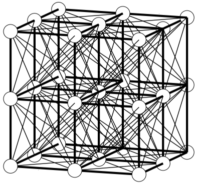
Figure 8: A 26-connected voxel adjacency graph.

Foveal sampling

"Space-variant sampling of visual input is ubiquitous in the higher vertebrate brain, because a large input space may be processed with high peak precision without requiring an unacceptably large brain mass." [1]

Figure 9: Some ducks. (Image from Grady 2004)

Foveal sampling

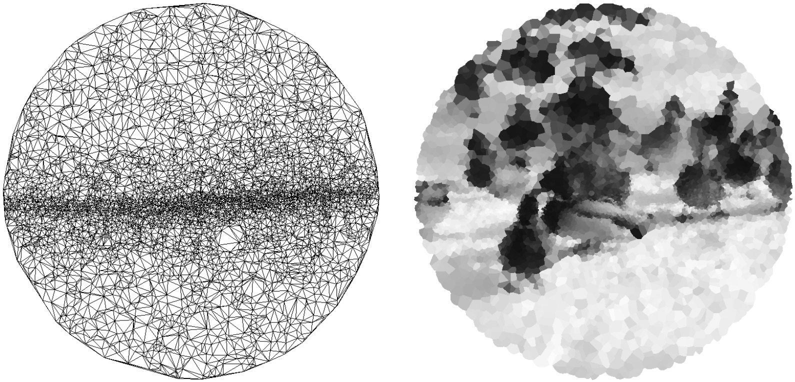
Figure 10: Left: Retinal topography of a Kangaroo. Right: Re-sampled image. (Images from Grady 2004)

Region adjacency graphs

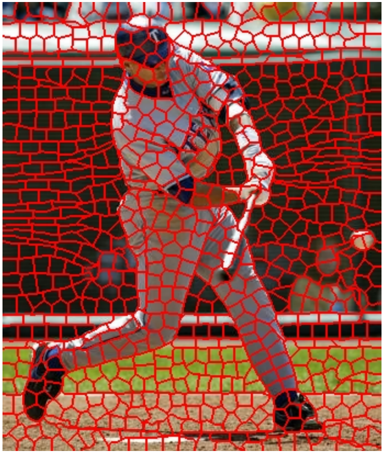

Figure 11: An image divided into superpixels

Multi-scale image representation

Resolution pyramids can be used to perform image analysis on multiple scales. Rather than treating the layers of this pyramid independently, we can represent the entire pyramid as a graph.

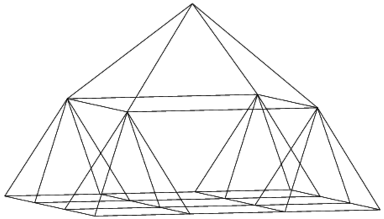

Figure 12: A pyramid graph (Grady 2004).

Directed and undirected graphs

- The pairs of vertices in E may be ordered or unordered.
  - In the former case, we say that G is directed.
  - In the latter case, we say that G is undirected.
- In this course, we will mainly (with some exceptions) consider undirected graphs.

Paths

- A *path* is an ordered sequence of vertices where each vertex is adjacent to the previous one.
- A path is *simple* if it has no repeated vertices.
- 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.

Commonly, simplicity of paths and cycles is implied, i.e., the word "simple" is ommited.

Example, Path

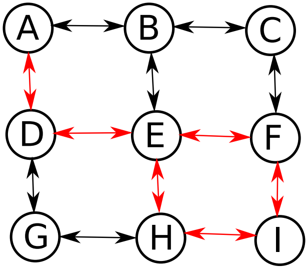

Figure 13: A path $\pi = \langle A, D, E, H, I, F, E \rangle$.

Example, Simple path

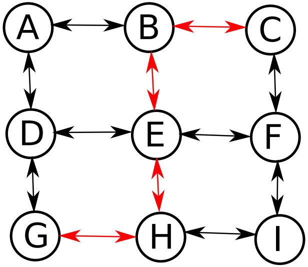

Figure 14: A simple path $\pi = \langle G, H, E, B, C \rangle$.

Example, Cycle

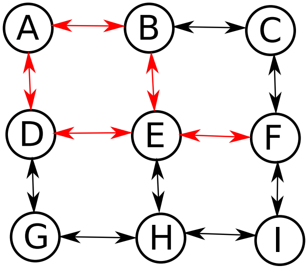

Figure 15: A cycle $\pi = \langle A, B, E, F, E, D, A \rangle$.

Example, Simple cycle

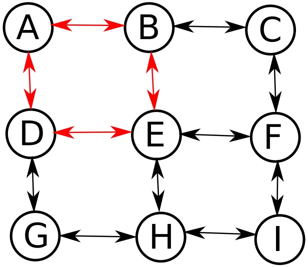

Figure 16: A simple cycle $\pi = \langle A, D, E, B, A \rangle$.

Paths and connectedness

- In an undirected graph, two vertices v and w are *connected* if there exists a path that starts at v and ends at w. We use the notation $v \underset{G}{\sim} w$. We can also say that w is *reachable* from v.
- If all vertices in a graph are connected, then the graph itself is said to be *connected*.

Subgraphs and connected components

- If G and H are graphs such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, then H is a *subgraph* of G.
- If H is a connected subgraph of G, and there are no paths in G connecting a vertex in H to a vertex not in H, then H is a *connected component* of G.

Example, connected components

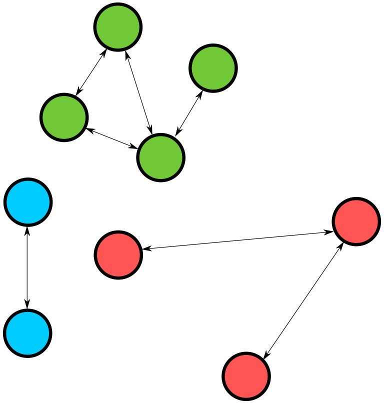

Figure 17: A graph with three connected components.

Implementing graph-based algorithms

- Even if we have formulated an algorithm on a general graphs, we do not neccesarily have to allow arbitrary graphs in *implementations* of the algorithm.
- For standard pixel/voxel adjacency graphs, we can evaluate adjacency relations without having to store the graph explicitly.

Implementing graph-based algorithms

If we do want to store the graph explicitly, there are some available libraries:

- For C++, I recommend the *Boost Graph* libraries. (www.boost.org)
- For Matlab, check out the *Graph Analysis Toolbox* (http://cns.bu.edu/ lgrady/software.html).
- For Python, there is, e.g., this library based on Boost: (https://graph-tool.skewed.de/)

References

L. Grady.
Space-Variant Machine Vision — A Graph Theoretic Approach.
PhD thesis, Boston University, 2004.