Interactive segmentation, Combinatorial optimization

Filip Malmberg

But first...

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).

Part 1: Interactive segmentation

Image segmentation

Wikipedia on segmentation:

- "In computer vision, Segmentation is the process of partitioning a digital image into multiple segments"
- "More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics."
- "Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic(s)."
- "Since there is no general solution to the image segmentation problem, these [general purpose] techniques often have to be combined with domain knowledge in order to effectively solve an image segmentation problem for a problem domain."

Image segmentation

Another view on segmentation:

- "Image segmentation is the task of partitioning an image into relevant objects and structures" Malmberg, 2011
- "Recent image segmentation approaches have provided ... methods that implicitly define the segmentation problem relative to a particular task of content localization. This approach to image segmentation requires user (or preprocessor) guidance of the segmentation algorithm to define the desired content to be extracted." Grady, 2006

Image segmentation

- Image segmentation is an ill-posed problem...

Figure 1? : What do we mean by a segmentation of this image?

Image segmentation

- Image segmentation is an ill-posed problem...
- ...Unless we specify a segmentation *target*.

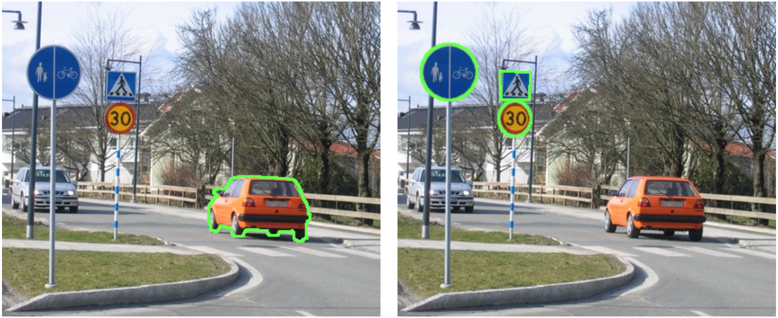

Figure 2? : Segmentation relative to semantically defined targets.

Image segmentation

We can divide the image segmentation problem into to two tasks:

- *Recognition* is the task of roughly determining where in the image an object is located.
  - High-level task.
  - Requires prior knowledge of the segmentation task/image.
- *Delineation* is the task of determining the exact extent of the object.
  - Low-level task
  - Can often be completed based on image data.

Semi-automatic segmentation

- Humans outperform computers in recognition.
- Computers outperform humans in delineation.
- *Semi-automatic* segmentation methods try to take advantage of this by letting humans perform recognition, while the computer does the delineation.
- The goal of semi-automatic segmentation is to minimize user interaction time, while maintaining a tight user control to ensure correct results.

Semi-automatic segmentation

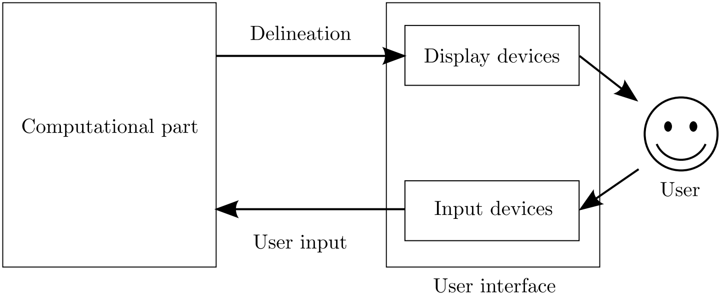

Figure 3? : The interactive segmentation process.

Paradigms for user input: Initialization

- The user provides an initial segmentation that is “close” to the desired one.

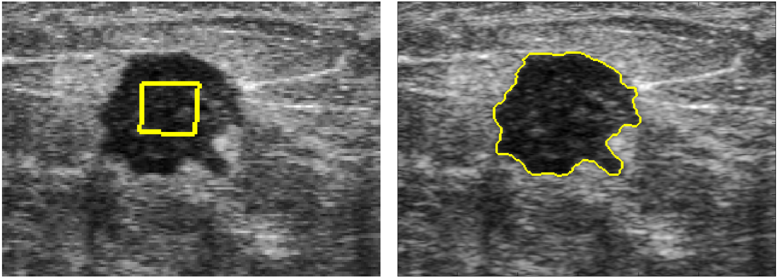

Figure 4? : Segmentation by initialization.

Paradigms for user input: Segmentation from a box

- The user is asked to provide a bounding box for the object

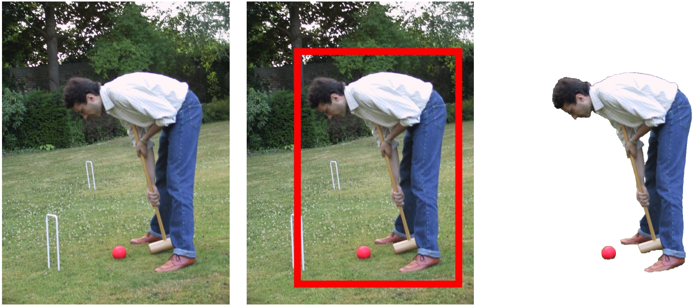

Figure 5? : Segmentation from a box.

Paradigms for user input: Boundary constraints

- The user is asked to provide points on the boundary of the desired object(s).

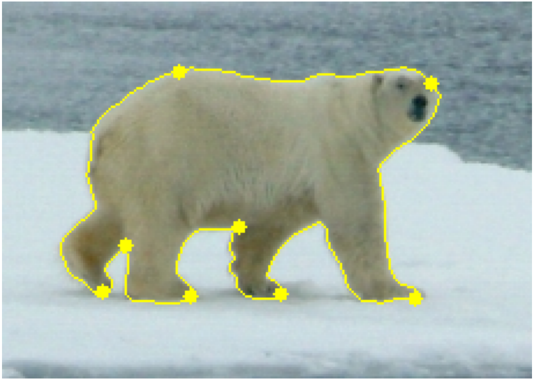

Figure 6? : Segmentation with boundary constraints.

Paradigms for user input: Regional constraints

- The user is asked to provide correct segmentation labels for a subset of the image elements ("seed-points")

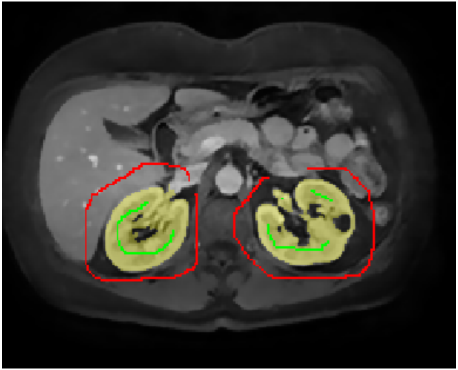

Figure 7? : Segmentation with regional constraints.

Hard and soft constraints

- The user input is commonly interpreted in one of two ways
  - *Hard constraints* - the conditions specified by the user must be satisfied exactly.
  - *Soft constraints* - the user input guides the segmentation algorithm towards a specific result, but does not reduce the set of feasible solutions.
- Hard constraints give a higher degree of control.
- Soft constraints may require less precise user input.

Desired properties of delineation methods

A delineation method ("computational part") takes an image, together with user input is some form, and produces a segmentation of the image. Desirable properties for a delineation method include:

- Fast computation.
- Fast editing.
- An ability to produce, with sufficient interaction, an arbitrary segmentation.
- "Intuitive" results.
- Robustness to "small" variations in user input.

Evaluation of interactive segmentation methods

- Evaluation of interactive segmentation methods differs slightly from evaluation of automatic segmentation methods.
- Segmentation methods can be evaluated according to:
  - Accuracy - how well the segmentation result corresponds to the "truth". We could argue that semi-automatic segmentation, by definition, is accurate.
  - Efficiency - How much time is required to obtain a segmentation result (user time/computer time).
  - Repeatability (precision) - How much does the result change if we repeat the segmentation (with slightly different input).

Part 2: Combinatorial optimization

Segmentation as an optimization problem

- We wish to find a cut/labeling that is *as good as possible* according to some criterion, while satisfying the constraints provided by the user.
- Typical measures of “goodness” may favour, e.g.:
  - Segmentations where object boundaries coincide with strong edges in the image.
  - Segmentation that divide the image into regions that are homogeneous with respect to some feature (intensity, color, texture).

Combinatorial optimization

- A combinatorial optimization problem consists of a *finite* set of candidate solutions $\mathcal{S}$ and an objective function $f : \mathcal{S} \to \mathbb{R}$.
- In segmentation, $\mathcal{S}$ could be the set of all possible vertex labelings (or cuts) of a graph.
- The objective function function f can measure either "goodness" or "badness" of a solution. Here, we assume that we want to find a solution $x \in \mathcal{S}$ that minimizes f.
- Ideally, we want to find a *globally minimal* solution, i.e., a solution $x^* \in \underset{x \in \mathcal{S}}{\operatorname{argmin}} f(x)$.

Combinatorial optimization

- It is tempting to view the objective function and the optimization method as completely independent. This would allow us to design an objective function (and a solution space) that describes the problem at hand, and apply general purpose optimization techniques.
- For an arbitrary objective function, finding a global optima recuires checking all solutions.
- The set $\mathcal{S}$ of solutions is finite. Can't we just search this set for the globally optimal solution?

How hard is combinatorial optimization?

- In vertex labeling, the number of possible solutions is $|L|^{|V|}$.
- Consider binary labeling of a 256×256 image.
- The number of possible solutions is 2^{65536}. This is a ridiculously large number!
- Searching the entire solution space for a global optimum is not feasible!

So, what do we do?

- For restricted classes of optimization problems, it is sometimes possible to design efficient algorithms that are guaranteed to find global optima. In upcoming lectures, we will cover some of these.
- Local search methods can be used to find *locally optimal solutions*. This is the topic of the remainder of this lecture.

Local optimality

- Define a neighborhood system $\mathcal{N}$ that specifies, for any candidate solution x, a set of *nearby* candidates $\mathcal{N}(x)$.
- A *local minimum* is a candidate x^* such that $f(x^*) \leq \min_{x \in \mathcal{N}(x^*)} f(x)$.

Local search

- A general method for finding local minima.
  - Start at an arbitrary solution.
  - While the current solution is not a local minimum, replace it with an adjacent solution for which f is lower.
- This algorithm is guaranteed to find a locally optimal solution in a finite number of iterations. *Why?*

Local search spaces as graphs

- We have a set $\mathcal{S}$ and an adjacency relation N.
- It's a (huge) graph!
- We never store this graph explicitly, but it can be useful to consider.
- For example, it seems reasonable to define the adjacency relation so that the graph of the search space is connected.

Local search

- "This algorithm is guaranteed to find a locally optimal solution in a finite number of iterations. Why?"
  - *If* the algorithm terminates, the result is a local minimum.
  - Each connected component in the graph of the search space contains at least one local minimum. (Why?)
  - The number of solutions is finite.
  - A solution is never visited more than once. (Why?)

Best-improvement search

- In *best-improvement search*, we consider *all* states in the local neighborhood of the current state. We accept the one that best improves the objective function.
- In *first-improvement search*, we consider the states in the local neighborhood of the current state one at a time. We accept the first one that improves upon the current state.
- Which one gives the best results? Which leads to a faster algorithm?

Extensions of local search

In standard local search, we accept *any* local minimum as a good solution. The following techniques modify the standard algorithm in an attempt to find "good" local minima.

- Local search with restarts
  - Run the algorithm several times, from different initial states. Select the best solution.
  - If an infinite number of restarts are allowed, a *global* optimum will be found with probability 1.
- Simulated annealing
  - Accept "worse" states with some probability, that may decrease over time.
  - Allows the algorithm to *explore* of harmful states, while *exploiting* successful states.

Local search, an example

Let's take a look simple binary thresholding

- Let $I(v)$ be the intensity of the pixel corresponding to v.
- Given a threshold t, we compute a vertex labeling according to:

$$\mathcal{L}(v) = \begin{cases} \textit{foreground} & \text{if } I(v) \geq t \\ \textit{background} & \text{otherwise} \end{cases} . \tag{1}$$

- Next, we will reformulate this as an optimization problem.

Local search, an example

We define the objective function f as

$$f = \sum_{v \in V} \Phi(v) \ , \tag{2}$$

where

$$\Phi(v) = \begin{cases} abs(max(t - I(v), 0)) & \text{if } \mathcal{L}(v) = foreground \\ abs(max(I(v) - t, 0)) & \text{otherwise} \end{cases} \ . \tag{3}$$

Local optimality, example

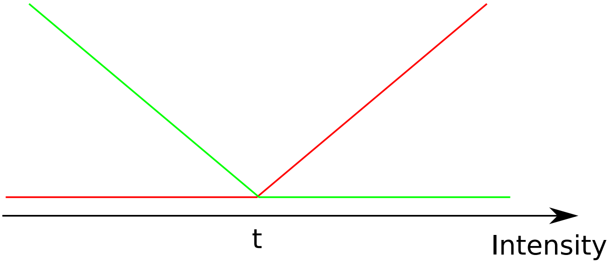

Figure 8? : Objective function for binary thresholding. The red curve is the cost of assigning the label “background” to a vertex with a certain intensity, and the green curve is the cost of assigning the “foreground” label.

Optimization by local search

- We say that two vertex labelings are adjacent if we can turn one into the other by changing the label of *one* vertex.
- We start from an arbitrary labeling, and use first-improvement search to find a locally optimal solution.

Local search, an example

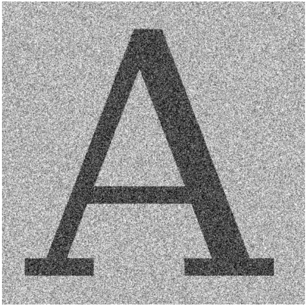

Figure 9? : Thresholding as an optimization problem.

Local search, an example

- Start from an arbitrary labeling.
- In this case, the label of each pixel does not depend on the label of any other pixels, so one iteration is sufficient.

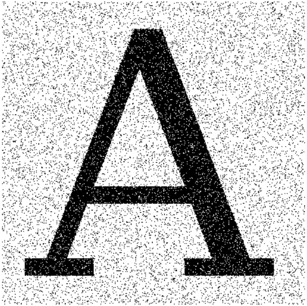

Figure 10? : Thresholding as an optimization problem.

Local search, an example

- We can add a term $|\partial\mathcal{L}|$, that penalizes long boundaries:

$$f = \sum_{v \in V} \Phi(v) + \alpha|\partial\mathcal{L}| \, , \tag{4}$$

where α is a real number that controls the degree of "smoothing".

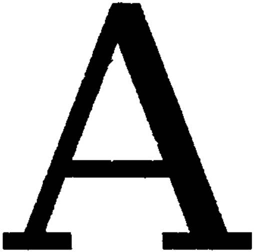

Figure 11? : Thresholding with smoothness term.

A note on efficient implementation

- In our example, the objective is a sum over all pixels in the image (and all edges in the cut corresponding to the current segmentation).
- Evaluating the entire objective function at each iteration is expensive.
- Instead, we can calculate how much the objective function *changes* when we change the label of a vertex.
- This is good to keep in mind when designing the objective function.

When is local search useful?

Similar solutions should have similar costs ("continuous" objective function).

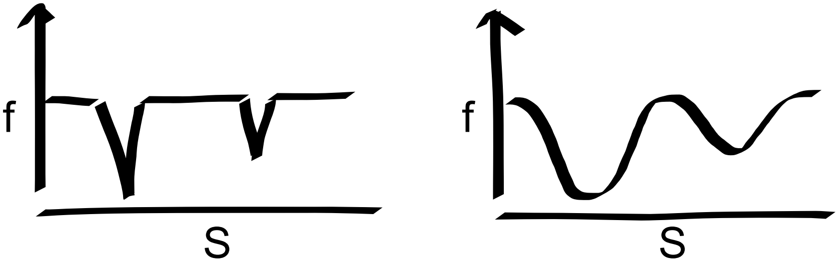

Figure 12? : (Left) An objective function that is hard to optimize using local search (Right) An objective function that is possible to optimize using local search.

Very large-scale neighborhood search

- To avoid getting trapped in poor local minima, it is desirable to use as large neighborhoods as possible.
- ...but large neighborhoods lead to slow computations.
- For some problems, we can find efficient algorithms for computing globally optimal solution within a subset of $\mathcal{S}$. If we use this subset as our local neighborhood, we can do best-improvement search!
- Erik Wernersson will talk about one such technique in his lecture.

Summary, interactive segmentation

- We define the segmentation problem relative to a given segmentation *task*.
- We can divide the segmentation problem into *recognition* and *delineation*.
- Interactive segmentation methods use human input to solve the recognition problem.
  - Many different paradigms for user input have been proposed.
- Evaluation of semi-automatic methods differs slightly from evaluation of automatic methods.

Summary, combinatorial optimization

- Many image processing problems, including segmentation, can be formulated as optimization problems.
- For arbitrary problems, finding a global optimum requires exhaustive search of the (huge) solution set.
    - For restricted classes of problems, we may be able to find global optima. (More on that in upcoming lectures!)
    - For other problems, we may be able to use hill-climbing techniques to find local optima.

Next lecture

- Optimal trees and forests.
- First example of a combinatorial optimization problem for which efficient global algorithms exist.