Combinatorial optimization and its applications in image Processing

Filip Malmberg

Part 1: Optimization in image processing

Optimization in image processing

Many image processing problems can be formulated as optimization problems - we define a function that assign a "goodness" value to every possible solution, and then seek a solution that is as "good" as possible.

- Image segmentation
- Image registrations/stereo matching/optical flow
- Image restoration/filtering

The "goodness" criterion is often referred to as an *objective function*.

Application 1: Image registration

Non-rigid Image registration

- Given two images, find a transformation (deformation field) that aligns one image to the other.
- Registration, stereo disparity, optical flow...

Example: Medical Image registration

Figure 1: Registering a series of whole body MRI images to match a common "mean person" facilitates direct comparisons between subjects.

Example: Optical flow

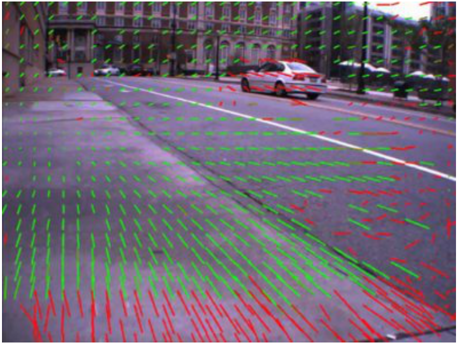

Figure 2: Registering consecutive frames in a video sequence facilitates, e.g., tracking and motion analysis.

Example: Optical flow for slowmotion

`https://www.youtube.com/watch?v=xLqz09-3vHg`

Figure 3: By computing optical flow in a video sequence, it is possible to interpolate new frames inbetween the captured ones, to simulte slow motion photography.

Image registration as optimization

- Image registration can be formulated as an optimization problem.
- Typically, we seek a solution that maximizes some notion of similarity between the images, while also maintaining some degree of smoothness of the deformation field.

Application 2: Image segmentation

Image segmentation

- Image segmentation is the task of partitioning an image into relevant objects and structures.
- Image segmentation is an ill-posed problem...

Figure 4: What do we mean by a segmentation of this image?

Image segmentation

- Image segmentation is the task of partitioning an image into relevant objects and structures.
- Image segmentation is an ill-posed problem...
- ...Unless we specify a segmentation *target*.

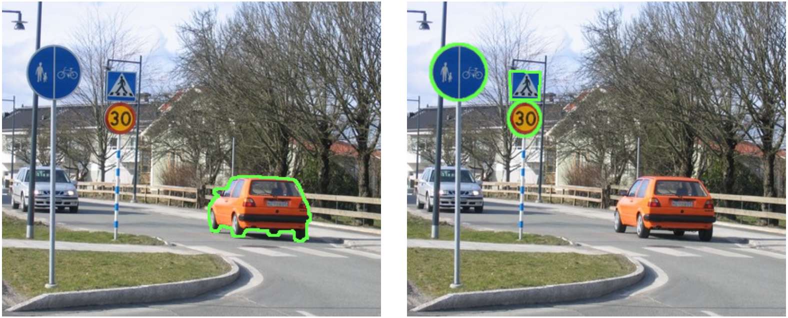

Figure 5: Segmentation relative to semantically defined targets.

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.
- *Delineation* is the task of determining the exact extent of the object.

Image segmentation

- *Recognition* information can be provided, e.g, by interactive annotations from a human user, or by an automatic algorithm incorporating high level knowledge.
- *Delineation* information can often be extracted from low-level image features. Typical such terms may favour:
  - Segmentations where object boundaries coincide with strong edges in the image.
  - Segmentations that divide the image into regions that are homogeneous with respect to some feature (intensity, color, texture).

Seeded segmentation / Semi-supervised learning

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

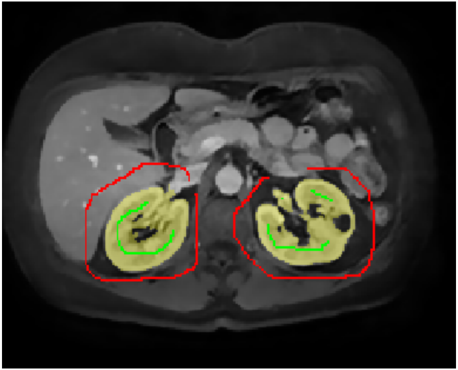

Figure 6: Segmentation with regional constraints.

Segmentation as an optimization problem

- We wish to find a segmentation that is *as good as possible* according to some criterion, based on recognition and delineation terms.

Application 3: Image restoration/filtering

Image filtering as an optimization problem

- Some image filtering operations can be formulated as an optimization problem with objective functions balancing two criteria:
  - The filtered image should be similar to the original one.
  - The filtered image should be smooth. (e.g. have small gradients)

Image filtering as an optimization problem, why?

- What do we gain from viewing filtering as an optimization problem?
- Perhaps not that much, for ordinary filtering operations.
- But it can be useful to keep this view if we want to develop new filters, e.g.,
  - edge preserving *anisotropic* filters
  - filters that "trust" some points in the data for than others

Example: Edge preserving filter

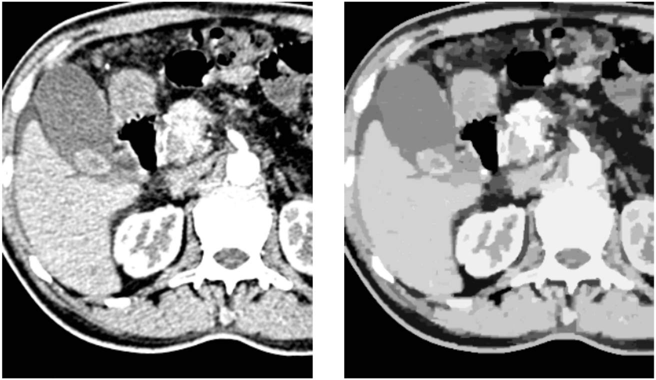

Figure 7: Image by Couprie et al.

"Typical" optimization problems in image analysis

The optimization problems occuring in the applications studied so far have a number of things in common:

- They are *pixel labeling* problems. In all cases, we seek to assign some type of *labels* (values) to the pixels of the image:
  - Object classes for segmentation.
  - Displacement vectors for registration.
  - Intensities/colours for restoration.
- The objective function consists of two terms:
  - A *data term* that measures how appropriate a label is for a certain pixel given some prior knowledge.
  - A *smoothness term* that favours spatial coherency.

Throughout the course, we will study optimization problems of this type.

Part 2: Combinatorial optimization

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 our examples, $\mathcal{S}$ will be typically be the set of all maps from the vertices of a graph to some set of *labels*.
- 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. (Proving this statement is one of the exercises!)
- Intuition: Local search $\approx$ "discrete gradient descent"

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?"
  - The number of solutions is finite.
  - *If* the algorithm terminates, the result is a local minimum. (Why?)
  - Each connected component in the graph of the search space contains at least one local minimum. (Why?)
  - 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?

Local search with restarts

- Run the algorithm several times, with different initialization.
- "Patience" factor. (Terminate local search if no local optimum find within k steps)
- With infinite patience, we will find a locally optimal solution with probability 1.
- With infinite restarts, we will find a globally optimal solution with probability 1.

Local search, an example

Let's take a look at pixel classification by 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:

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

- 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

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

Local search, example

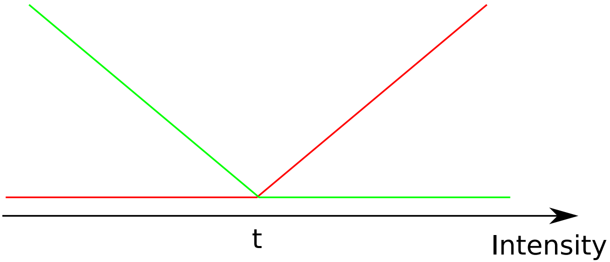

Figure 8: Objective function for binary thresholding. The red curve is the cost of assigning the label “background” to a vertex with a certain intensity, and the green curve is the cost of assigning the “foreground” label.

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.

Optimization by local search, algorithm

```
done=false
while done do
    done=true
    foreach pixel p in the image do
        Can we improve the current solution by changing the label of p?
        If so, change the label and set done=false.
    end
end
```

Local search, an example

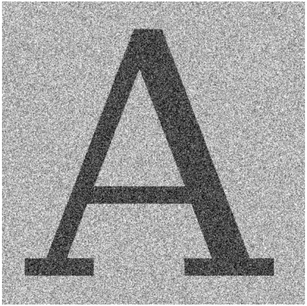

Figure 9: Thresholding as an optimization problem.

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 a local optimum is reached after only one iteration of the while-loop.
- (This optimum is in fact also global)

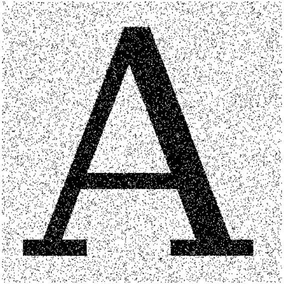

Figure 10: Thresholding as an optimization problem.

Local search, an example

- Let us add to the objective function a smoothness 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".

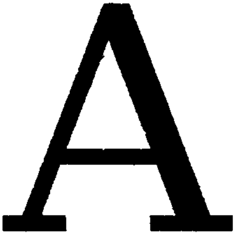

Figure 11: Thresholding with smoothness term.

Unary and binary terms

- The data term ϕ in the example is a sum over the pixels in the image. In this term, each pixel is considered individually. We say that ϕ is a *unary* term.
- In contrast, the smoothness term is defined over all *pairs* of adjacent pixels (edges, in the graph context). We say that this term is *binary*.

Local search, an example

- After adding the (binary) smoothness term, we have introduced a dependency between the labels of adjacent pixels. We can no longer decide on the best label for each pixel independently!
- This makes the optimization problem harder to solve.
  - The local search algorithm requires many passes over the image before convergence.
  - The local solution is no longer guaranteed to be a global optimum.

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).

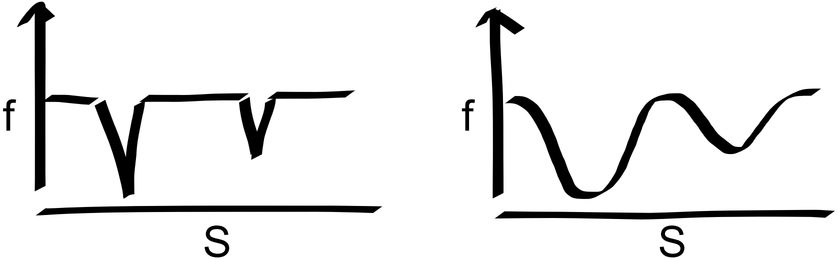

Figure 12: (Left) An objective function that is hard to optimize using local search (Right) An objective function that is possible to optimize using local search.

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!
- We will look at one such technique in an upcoming lecture.

Global optimization

- For a general combinatorial optimization problem, finding the optimal solution requires checking all solutions.
- For specific classes of problems, we can do better!
- Quite remarkably, there are many algorithms for solving optimization problems of interest in image analysis that guarantee globally optimal results
- In this course we will cover some of the most important such methods.

Summary

- Many image analysis problems can be formulated as (combinatorial) optimization problems.
- Local search methods can be used to find *locally* optimal solutions to any combinatorial optimization problem.
- Depending on the problem and the local search strategy used, these locally optimal solutions may or may not be good enough. (But often we can use methods that produce really "good" local optima!)
- For many interesting combinatorial optimization problems we can find globally optimal solutions efficiently. More on this later!

Reading material for next time

- Topic: “Optimal spanning forests, shortest paths and Dijkstra’s algorithm”
- Paper 1: “The Image Foresting Transform: Theory, Algorithms, and Applications”, IEEE PAMI 2004
- Paper 2: “Path-Value Functions for Which Dijkstra’s Algorithm Returns Optimal Mapping”, JMIV 2018
- Both papers are avaliable on the course webpage