Computer vision: Foundations
Summer Semester 2020

Exercise Sheet #6

Due: 16.06.2020, 15:15

Fred Hamprecht, Alberto Bailoni
`cvflecture.s20@gmail.com`

General Regulations

- No individual submission! Hand in your solutions in groups of two or three people (preferred).
- If possible, theoretical exercises should be typeset, either by submitting a separate `.pdf` file created with LaTeXor by providing your answers in a jupyter notebook (use markdown blocks and LaTeX code for equations). Alternatively, a scan of handwritten notes is also accepted.
- Submit all your files in a single `.zip` archive (not `.rar` or `.tar`) and upload it using the Physics platform `https://uebungen.physik.uni-heidelberg.de/h/1176`. **Only one person in the group should upload the zip archive**: in the upload interface you will see the option to indicate your exercise-partners.

1 Dijkstra Algorithm (5pt)

In this exercise, we consider the simple weighted graph $\mathcal{G}(V, E, w)$ shown in Fig. 1 and compute the shortest path using the Dijkstra Algorithm 1.

(a) (1pt) First, have a look at Dijkstra Algorithm 1: What is the returned length of the shortest path between the source s and a node $v \in V$, if there is no path connecting s and v in the graph?

(b) (1pt) If we use the Dijkstra algorithm on a graph with negative loops (i.e. loops where the sum of the edge weights is negative), does the algorithm work? Explain your answer.

(c) (2pt) Now consider the graph in Fig. 1 and apply the Dijkstra Algorithm. Complete Table 1 with all the iterations performed by the algorithm (no need to implement the algorithm, you can solve this small problem with pen and paper). The i-th row of the Table reports the node selected at iteration i (first column) and the content of the `distance` and `prevNode` vectors (remaining columns). For a given column and associated node v, use the subscript notation $\texttt{distance}[v]_{\texttt{prevNode}[v]}$ as shown for the first iteration in Table 1.

(d) (1pt) What is the shortest path from node A to node G? Write it down as a sorted list of nodes belonging to it. What is its length?

Algorithm 1 Shortest path: Dijkstra Algorithm

Inputs: Weighted graph $\mathcal{G}(V, E, w)$ with nodes V, edges $E \subseteq V \times V$ and weights $w : E \to \mathbb{R}$; source node s.

```
1: for $v \in V$ do
2:     distance[$v$] $\leftarrow +\infty$          ▷ Initialize distance from source $s$ to vertex $v$
3:     prevNode[$v$] $\leftarrow$ None             ▷ Initialize previous node in optimal path from source
4: distance[$s$] $\leftarrow 0$                    ▷ Distance from source to source is zero
5: $Q \leftarrow V$                                ▷ Insert all nodes into the queue
6: while $Q$ is not empty do                       ▷ Loop over all not processed nodes
7:     $u \leftarrow \arg\min_{v \in Q}$ distance[$v$]
8:     $Q \leftarrow Q \setminus \{u\}$
9:     for $(u, v) \in \{(u, v) \in E | v \in Q\}$ do
10:        if distance[$u$] + $w_{uv}$ < distance[$v$] then
11:            distance[$v$] $\leftarrow$ distance[$u$] + $w_{uv}$
12:            prevNode[$v$] $\leftarrow u$
13: return (distance, prevNode)
```

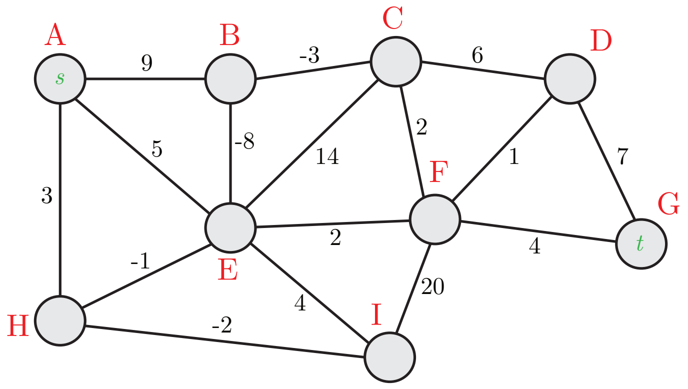

Figure 1: Simple weighted graph $\mathcal{G}(V, E)$. The source and target are given by nodes A and G, respectively.

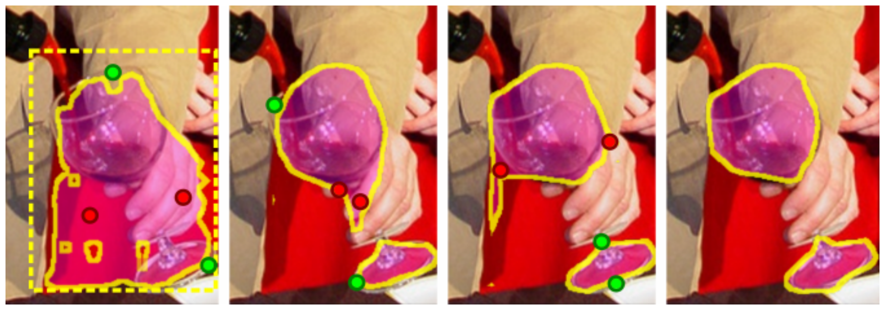

Figure 2: Example of corrective clicks/strokes and their effect on the segmentation mask. Red clicks indicate background, whereas green ones indicate foreground. Each time the annotator provides some new clicks/strokes, the inference process is repeated and the new improved segmentation result is shown. The correction process is repeated for several rounds until a final satisfactory mask is obtained. This explanatory picture was taken from a recently proposed method (*Benenson Rodrigo et al. "Large-scale interactive object segmentation with human annotators." CVPR 2019.*)

2 Shortest path and interactive instance segmentation (7pt)

In the first exercise sheet, we have used interactive learning in ilastik to perform a semantic segmentation of yeast-cells. In this exercise, we will see how to integrate what we learned about the shortest path problem into a possible interactive work-flow for object segmentation. In the example work-flow shown in Fig. 2, an annotator provides clicks/strokes indicating background/foreground pixels at different rounds. At each round, the inference process (which in our case will be given by solving a shortest path problem) is repeated to output an updated segmentation mask. The correction process is repeated for several rounds until a final satisfactory mask is obtained.

(a) Graph and problem definition (2pt) We now represent the input image as a weighted pixel grid-graph $\mathcal{G}(V, E, w)$, such that each vertex in the graph represents a pixel of the image and neighboring edges are connected by weighted edges (each pixel has four neighbors). Assume that the graph $\mathcal{G}(V, E, w)$, its weights $w_e : E \to \mathbb{R}$ and some background/foreground annotations from the annotator user are given. How would you output the final object mask by solving a shortest path problem? Describe the graph you would use and how the algorithm would proceed.

(b) Precomputed features (2pt) How should the edge weights of the graph look like in order to obtain a reasonable object segmentation mask with the shortest path algorithm you described in the previous task? Without using any machine learning algorithm, how would you compute these edge weights?

(c) Deep-learning based model (3pt) Now consider what you have learned in the last weeks about deep learning models for semantic and instance segmentation. What would be a way to compute better edge weights and obtain a more accurate segmentation mask with the shortest path algorithm? Assume that you have hardware to train a deep learning model and that you have training data available (for example from the CityScapes dataset instance segmentation task introduced in the last exercise sheet). Which model would you use and how would you train it? What would the outputs of your model be? *Hint: keep in mind that training a deep learning model requires time and a lot of training data. On the other hand, getting the predictions of the deep-learning model for one single image does require very little time.*

3 Bonus: Maximum subarray problem (4pt bonus)

The maximum subarray problem[1] is the task of finding a contiguous subarray with the largest sum, within a given one-dimensional array $A[1...n]$ of positive and negative numbers. In formulas, the task is to find indices i and j with $1 \leq i \leq j \leq n$ such that the sum

$$\sum_{x=i}^{j} A[x] \tag{1}$$

is as large as possible. Note that that an empty subarray can also be considered (and the sum of the elements in an empty subarray is zero). In this exercise, we will reformulate this problem as a *Longest path problem* on a directed acyclic graph.

(a) Longest path problem (1pt) Given a weighted graph $\mathcal{G}(V, E, w)$, the *longest path problem* is the problem of finding a simple path of maximum length. A path is called *simple* if it does not have any repeated vertices and the length of a path is defined as the sum of the weights of its edges. Now consider the small graph in Fig. 1 and find the longest path from node A to G. Report which nodes are included in the path and its length.

(b) Maximum subarray problem (3pt) As we have seen, the *shortest path problem* can be solved in polynomial time in graphs without negative-weight cycles. Despite the fact that *shortest* and *longest path problems* sound very similar, it turns out that the *longest path problem* is NP-hard. However, it has a linear time solution for directed acyclic graphs[2]. Can you think of a way to reformulate the *maximum subarray problem* (for 1D arrays) as a *longest path problem*? *Hint: If the array has length n, then the associated graph should have $n + 2$ nodes.*

[1] Definition of maximum subarray problem on Wikipedia: `https://en.wikipedia.org/wiki/Maximum_subarray_problem`

[2] The reason for this is the following. Consider a graph G and its negation graph $-G$ derived by changing every weight to its negation. If G is directed and acyclic, then the negation graph $-G$ is also directed and acyclic (and it does not include any negative-weighted cycle). Thus, the longest path in G can be found by applying an efficient algorithm for shortest paths in $-G$.

Iteration number	Selected node with minimum distance	A	B	C	D	E	F	G	H	I
-	-	$\boxed{0}$	∞	∞	∞	∞	∞	∞	∞	∞
1	A		9_{A}	∞	∞	5_{A}	∞	∞	$\boxed{3_{\mathrm{A}}}$	∞
2	.	.	.	.	.	.	.	.	.	.
3	.	.	.	.	.	.	.	.	.	.
4	.	.	.	.	.	.	.	.	.	.
5	.	.	.	.	.	.	.	.	.	.
6	.	.	.	.	.	.	.	.	.	.
7	.	.	.	.	.	.	.	.	.	.
8	.	.	.	.	.	.	.	.	.	.

Table 1: Table representing the iterations of the Dijkstra Algorithm on the graph in Fig. 1.