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).
- All outputs, in particular images, should be visible in the notebook. We should not have to run your notebook or uncomment lines / change some code to see your results!
- When asked, remember to comment / interpret your results.
- 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.

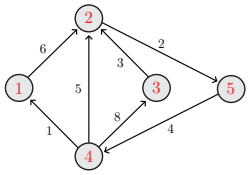


Figure 1

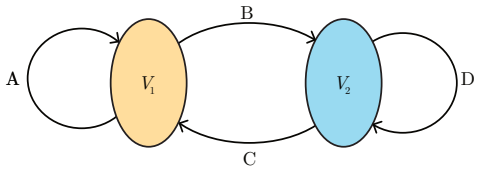


Figure 2: Graph G and the two sets of nodes V_1, V_2

1 Generalized All-Pairs Shortest Path Problem (6pt+2pt bonus)

(a) All-pairs shortest path (3pt) Compute the length of the shortest path for all pairs of nodes in the small directed graph of Fig. 1 by using the matrix-multiplication method that you have seen during the lecture. You can do it manually or by implementing it in python. *Hint: you only need two matrix-multiplication operations.*

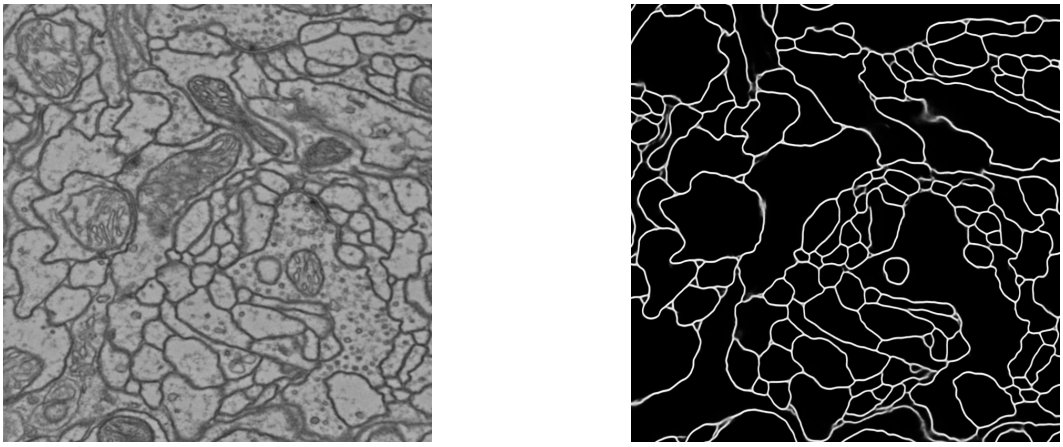
(b) All-pairs minimax path (2pt) Now re-do the previous computations to find the length of the minimax path between any pair of nodes, i.e. the path that minimizes the maximum weight of any of its edges.

(c) Negative cycles (1pt) How would you detect the presence of negative cycles in a graph by slightly modifying the algorithm/code that you used to solve the previous tasks?

(d) Matrix property (2pt bonus) Let us consider a directed graph G with matrix of edge weights X. For any matrix A that represents edge weights, we define A^* as the matrix that holds all the shortest path lengths (like the one you computed in task (a)). Given two matrices A and B, let AB denote the min-plus product of A and B, and let $A \wedge B = \min\{A, B\}$ be the element-wise minimum of the two matrices. Show and explain why the following statement holds:

$$\text{If } X := \begin{pmatrix} A & B \\ C & D \end{pmatrix} \text{ and } X^* := \begin{pmatrix} E & F \\ G & H \end{pmatrix} \implies E = (A \wedge (BD^*C))^* \tag{1}$$

See Fig. 2 for a schematic representation of the graph associated to X. The statement in Eq. 1 (together with other properties of the matrix X^* that are not shown here) can be used to prove that in most cases the cost of solving the all-pairs shortest path problem on graphs with n vertices is equal to $\mathcal{O}(MPP(n))$, where $MPP(n)$ is the cost of computing the min-plus product of two $n \times n$ matrices (but you do not need to show this in this exercise).



(a) Raw image (b) Boundaries predicted by a UNet model

Figure 3: Crop of the CREMI dataset (electron microscopy images of Drosophila melanogaster brain tissue)

2 Distance Transform Watershed Using Fiji (4pt+2pt bonus)

ImageJ[1] is a user-friendly and open source program for image processing, which has a strong and established community and includes thousands of plugins for performing a wide variety of tasks. In this exercise, you will use the software to apply the Distance Transform Watershed to the CREMI dataset that we introduced in the previous assignment. This method is commonly used to generate an over-segmentation on this type of data.

Installing Fiji – Fiji[2] is a distribution of ImageJ that comes with a lot of plugins already installed. Download and install Fiji and then have a look at this tutorial[3] on how to use the Distance Transform Watershed function: you will first need to follow the instructions to install the extra *MorphoLibJ* library. In the main window of Fiji (in the bottom right corner), you find a *Search* field that let you search commands in the menus (useful since there are many). Here there is a list of commands/tools that you may find useful to solve the next tasks: *Threshold, Distance Map, Distance Transform Watershed, Save As.*

(a) Distance Transform (2pt) Load the provided image `boundary_predictions.png` (shown in Fig. 3) into Fiji and compute a boundary distance-transform: the value of each pixel in the output image should represent the distance to the closest boundary (which is zero if the pixel was classified as boundary). Save the image as `distance-transform.png`.

(b) Distance Transform Watershed (2pt) Now follow the tutorial[3] to apply the Distance Transform Function. Save the image as `distance-transform-WS.tif` and comment your results. If you want, you can use the provided code in *ex09.ipynb* to overlay the obtained segmentation with the original raw image. *Hint: you may need to convert the final segmentation from float32 to int16 before to export it (Image/Type/16-bit).*

(c) Segmenting boundaries (2pt bonus) You may have noted that in the previous task boundaries are considered as background-pixels. However, in this type of data *every* pixel should be assigned to a neuron/segment. What method would you use to fix this problem? Use some of the other tools available in Fiji to test your idea and then save the output segmentation as `distance-transform-WS-no-boundaries.tif`.

[1] Website of ImageJ: https://imagej.net/ImageJ.
[2] Website of Fiji: https://imagej.net/Fiji
[3] Tutorial on Distance Transform Watershed: https://imagej.net/Distance_Transform_Watershed

3 Bonus: Labeling Connected Components (6pt bonus)

In this exercise you will have a look at an efficient connected component labeling algorithm, which is used in computer vision to detect connected regions in binary images. In particular, we will focus on the efficient *Two-pass* algorithm: you can read more about it and its pseudocode on the associated wikipedia page[4]. At this webpage[5] you can also find a nicely explained example of how the algorithm proceeds.

(a) Smallest example with three redirections (3pt) Assume to have an image with only one connected component. After running the first pass of the algorithm, the resulting label image has three different labels. This means that during the first pass the algorithm called the `union()` method of the union-find data structure at least two times (so that in the second pass all three labels are overwritten by a single final label). Find the smallest binary image for which this is happening. Note that we consider a 4-neighbors connectivity and that the algorithm analyzes one row after the other.

(b) Connected Components on Graphs (3pt) Given an undirected graph $G = (V, E)$, we say that two vertices are *connected* if it exists a path in the graph that connects them. A connected component of an undirected graph $G = (V, E)$ is a subgraph $G' = (V' \subseteq V, E' \subseteq E)$ in which each vertex $v' \in V'$ is connected only to the other vertices in V', but not to the vertices in $V \setminus V'$. The *Two-Pass* algorithm you have seen in task (a) works for images or more generally for graphs with a regular grid-structure. How would you approach the problem on a general graph? Keep in mind what you learned in the last assignment about the UnionFind data structure (Sheet 8, Exercise 1) and use the code provided in the `ex09.ipynb` notebook to implement a connected component algorithm for general graphs. Then test your idea on a randomly generated binomial graph (follow the instructions provided in the `ex09.ipynb` notebook).

[4] Wikipedia page of the connected component labeling problem: https://en.wikipedia.org/wiki/Connected-component_labeling

[5] Webpage with a nice example of how the two-pass connected component algorithm proceeds: https://aishack.in/tutorials/labelling-connected-components-example