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).
- Only include a single jupyter notebook per submission. Not one per person or separate ones per exercise.
- 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.
- Every result or image you provide should be at least pointed to in the notebook (by file / directory name).
- Include all files/code that include your work in the zip file. Otherwise, it will receive no marks.
- 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 Build a maze with Kruskal's algorithm (5pt)

During the lecture, we have seen Prim's algorithm for computing a minimum spanning tree. In this exercise, we will explore another one, Kruskal's algorithm, and use it to build a maze. The pseudocode of the algorithm and a small example can be found on the algorithm's Wikipedia page[1]. While implementing the algorithm, you will get to work with the Disjoint-set data structure (also called *union-find data structure*), which often plays an important role in computer vision because it provides a super efficient way to check whether two nodes are part of the same cluster or not. You can read more about it on the related Wikipedia page[2].

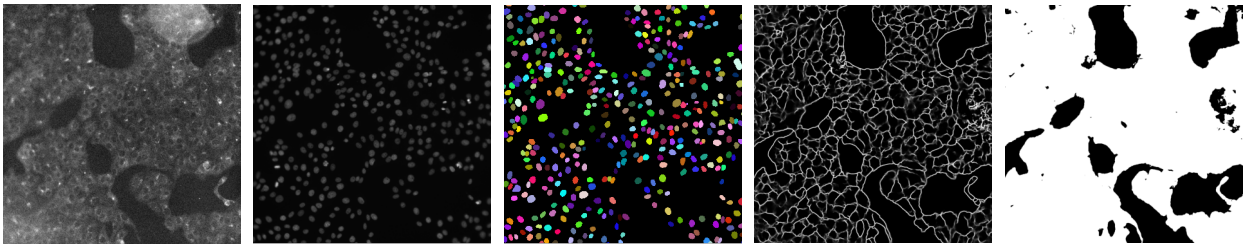
Building a maze A maze can be generated by starting with cells (arranged in a grid) with wall sites between them. This arrangement can be considered as a connected graph with the edges representing possible wall sites and the nodes representing cells. The purpose of the maze generation algorithm can then be considered to be making a subgraph (selecting a subset of edges) in which it is challenging to find a route between two particular nodes.

We can start in a graph without any edges (all walls are erect) and procedurally add edges (tear down a wall) until we have generated a maze. Our main concern is to generate a maze that has a connection between start and end point. One way to ensure this property is to generate a spanning tree over the graph which connects every pair of points with a unique path.

(a) Implement Kruskal's algorithm (4pt) Generate a random (16×16) maze with an entrance at $(0, 0)$ and an exit at $(15, 15)$ using Kruskal's algorithm. To help you along, in `ex08.ipynb` you find a partial implementation and a method to visualize the constructed maze. Finish the implementation of Kruskal's algorithm and plot the resulting maze. A python implementation of the union-find data structure can be found in `cvf20/unionfind.py`.

(b) Build an unsolvable maze (1pt) Now slightly modify your code to make sure that the entrance of the maze at $(0, 0)$ and the exit at $(15, 15)$ are no longer connected by a path. *Hint: given the planar*

[1] Wikipedia page of Kruskal's algorithm: https://en.wikipedia.org/wiki/Kruskal%27s_algorithm
[2] Wikipedia page of the Disjoint-set data structure: https://en.wikipedia.org/wiki/Disjoint-set_data_structure



(a) IgA channel (b) Nuclei channel (c) Segmented Nuclei (d) Cell Boundaries (e) Foreground

Figure 1: Example of data from the study [1] detecting Sars-CoV-2 specific antibodies. In (a) and (b) you see two channels of the raw input (IgA and nulcei channel). The nuclei segmentation in (c) is obtained using the StarDist method [2], whereas cell boundaries in (d) and the foreground prediction in (e) are obtained by using a U-Net model. See the original paper for more details.

grid-graph describing the maze, the output of the algorithm should be a spanning forest made of two trees.

2 Cell Segmentation with the Watershed Algorithm (4pt)

If you completed the previous exercise, then you probably realized that you have successfully implemented something very similar to a seeded Watershed algorithm! In this exercise, you will apply a more efficient implementation of the Watershed algorithm to the data that was briefly presented during the lecture and that was published together with a very recent study [1] introducing a method to detect the three major classes of Sars-CoV-2 specific antibodies (IgG, IgA and IgM) from microscopy-based screens of human blood samples.

In Fig. 1 you see an example of the data that you will use in this exercise and that includes cell boundary predictions, nuclei instance segmentation and foreground/background predictions. For an overview of the full image-analysis workflow proposed in the study, see Fig. 2 in the original paper [1].

(a) Watershed (2pt) Read the documentation of the Watershed algorithm implementation in the `scikit` module. Then use the cell boundary predictions in Fig. 1d to output the watershed segmentation, where seeds are generated from local-minima. Plot and comment your results using the provided code in `ex08.ipynb`. *Hint: the* `skimage.segmentation.watershed` *function has the possibility to mask the background pixels.*

(b) Seeded Watershed (2pt) Now use the nuclei segmentation in Fig. 1c as seeds for the watershed algorithm. Plot and comment your results.

3 Bonus: Neuron Segmentation with Mutex Watershed (5pt bonus)

If it is not possible to automatically generate good seeds like in Exercise 2, then it becomes more difficult to obtain decent segmentation results with just a Watershed algorithm. A possible way to eliminate the need for seeds is given by the Mutex Watershed Algorithm [3], which is an efficient and simple algorithm that can partition graphs with both positive and negative weights, where positive weights represent an *attraction* between nodes that would like to be in the same instance, whereas negative weights represent repulsive interactions between nodes in the graph.

In Algorithm 1 you can review the pseudo code of the MWS algorithm (see Fig. 3 in the original paper[3] for an example of how the algorithm operates on a small toy graph). Similarly to the algorithms you have seen in the previous exercises, the MWS also relies on an union-find data structure to keep track of all

[3]You can find an updated version of the paper provided with the assignment material (`Mutex Watershed paper.pdf`).

clusters during the merging progress. It then uses sorted lists to keep track of all active mutual exclusion constraints.

In this exercise, we will use the Mutex Watershed (MWS) Algorithm to find an instance segmentation of fruit fly brain tissue like the one in Fig. 2, without the need of seeds. The positive and negative edge weights of the pixel grid-graph are derived from the predictions of an edge detector, given by a convolutional neural network.

(a) Mutex Watershed Algorithm (5pt bonus) Complete the partial MWS implementation given in `cfv20/mws_python.py` and then use the code in `ex08.ipynb` to test it on a graph generated from an image of fruit fly brain tissue. Visualize the output neuron segmentation and see how it compares to the provided ground truth labels.

Mutex Watershed:
MWS$\big(\mathcal{G}(V,E)$**,** $w : E \to \mathbb{R}$**:**

```
A⁺ ← ∅;   A⁻ ← ∅;
for (i, j) = e ∈ E in descending order of |w_e| do
    if w_e > 0 then
        if not mutex(i, j; A⁺, A⁻) then
            if not connected(i, j; A⁺) then
                merge(i, j): A⁺ ← A⁺ ∪ e;
                  ▷ merge i and j and inherit the mutex
                    constraints of the parent clusters
            end
        end
    else
        if not connected(i, j; A⁺) then
            addmutex(i, j): A⁻ ← A⁻ ∪ e;
                  ▷ add mutex constraint between i and j
        end
    end
end
return A⁺ ∪ A⁻
```

Algorithm 1: Mutex Watershed Algorithm. The output clustering is defined by the connected components of the final spanning forest A^+. The connected predicate can be efficiently evaluated using union find data structures.

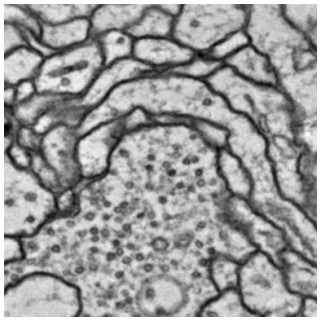


Figure 2: Electron microscopy data of adult Drosophila melanogaster brain tissue. In neuron segmentation, the task is to segment each neuron with a different ID. In other words, we need to solve an instance segmentation problem where each pixel in the image should be assigned to one single neuron/instance.

References

[1] C. Pape, R. Remme, A. Wolny, S. Olberg, S. Wolf, L. Cerrone, M. Cortese, S. Klaus, B. Lucic, S. Ullrich, et al. Microscopy-based assay for semi-quantitative detection of sars-cov-2 specific antibodies in human sera. *bioRxiv*, 2020.

[2] U. Schmidt, M. Weigert, C. Broaddus, and G. Myers. Cell detection with star-convex polygons. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 265–273. Springer, 2018.

[3] S. Wolf, A. Bailoni, C. Pape, N. Rahaman, A. Kreshuk, U. Köthe, and F. A. Hamprecht. The mutex watershed and its objective: Efficient, parameter-free graph partitioning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.