Technical Report

Pattern Recognition and Image Processing Group
Institute of Visual Computing and Human-Centered Technology
TU Wien
Favoritenstrasse 9-11/193-03
A-1040 Vienna AUSTRIA
Phone: +43 (1) 58801 - 18661
Fax: +43 (1) 58801 - 18697
E-mail: carmine_carmine@libero.it
URL: http://www.prip.tuwien.ac.at/

PRIP-TR-153 May 13, 2022

Distance transform of generalized maps and applications to gas exchange analysis

Carmine Carratù

Abstract

The distance transform computes for every pixel of a binary image the distance from the closest pixel in the background. The operator can be generalized to work with other data structures like graphs or combinatorial maps. In this work we describe how to compute the distance transform of a generalized map, and we propose two algorithms to compute respectively the distance transform of unweighted and weighted generalized maps. Moreover, we describe how such algorithms can be used for the "Water's gateway to heaven" project[1] to compute the diffusion distance and Voronoi diagrams that can be used to study the gas exchange process that occurs in plant leaves.

Contents

1 Introduction and state of the art analysis

1.1 Generalized maps

We will give a brief introduction to generalized maps, introducing the data structure and the basic operations needed to understand this work. A detailed discussion on the topic can be found in [2], [3], [4]. The content of this chapter, including the images, is mainly derived from the sources cited above, especially from [2].

An n-Gmap is a combinatorial data structure allowing to describe an n-dimensional orientable or non-orientable quasi-manifold with or without boundaries. An n-Gmap is defined by a set of darts on which act n + 1 involutions, satisfying some composition constraints. This leads to the following definition of n-Gmaps.

n-Gmap definition An n-dimensional generalized map, or n-Gmap, with $0 \leq n$ is an (n+2)-tuple $G = (D, \alpha_0, ..., \alpha_n)$ where:

1. D is a finite set of darts,
2. $\forall i \in \{0, ..., n\} : \alpha_i$ is an involution on D
3. $\forall i \in \{0, ..., n-2\}, \forall j \in \{i+2, ..., n\} : \alpha_i \circ \alpha_j$ is an involution.

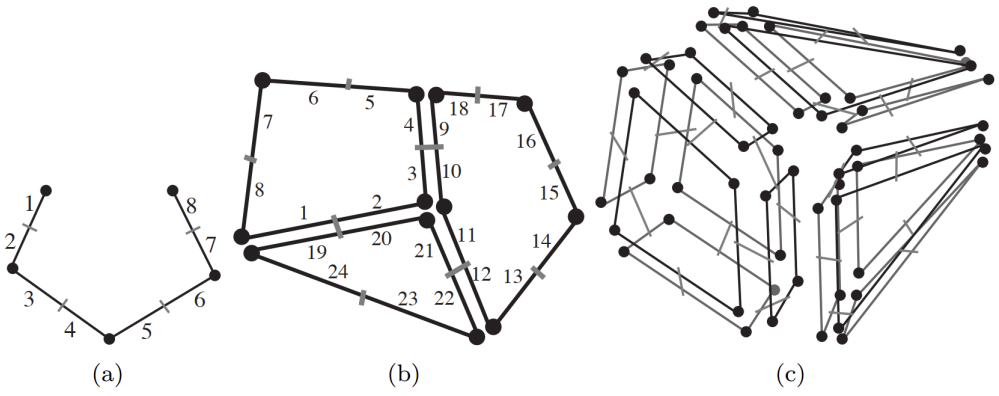

Figure 1: Examples of (a) 1-Gmap, (b) 2-Gmap and (c) 3-Gmap

A 0-Gmap (D, α_0) represents the structure (the topology) of a set of isolated vertices, or pairs of vertices (corresponding thus to 0-spheres); a 1-Gmap (D, α_0, α_1) represents the structure of a set of polygonal curves with or without boundaries Fig. 1 (a); a 2-Gmap $(D, \alpha_0, \alpha_1, \alpha_2)$ represents the structure of a set of surfaces Fig. 1 (b); a 3-Gmap $(D, \alpha_0, \alpha_1, \alpha_2, \alpha_3)$ represents the structure of assemblies of volumes Fig. 1 (c) ...

As we said darts represent the basic elements of an n-Gmap. They are linked each other by involutory functions. For example in Fig. 1 (b) $\alpha_0(4) = 3; \alpha_1(4) = 5; \alpha_2(4) = 9$. Moreover, since all the functions are involutions, it means:

if $\alpha_i(d) = \alpha_i(d')$ then $\alpha_i(d\prime) = \alpha_i(d)$ $\forall d \in D, \forall i \in \{0, ..., n+1\}$

i-cell definition Let $G = (D, \alpha_0, ..., \alpha_n)$ be an n-Gmap, $d \in D$, and $i \in \{0, ..., n\}$. The i-dimensional cell (or i-cell) containing d is:

$$c_i(d) = \langle \alpha_0, ..., \alpha_{i-1}, \alpha_{i+1}, ..., \alpha_n \rangle (d) \tag{1}$$

So for example in Fig. 1 (b) the set of darts $\{2, 3, 10, 11, 21, 20\}$ represents a 0-cell (vertex), the set $\{1, 2, 20, 19\}$ represents a 1-cell (edge) and the set $\{1, 2, 3, 4, 5, 6, 7, 8\}$ represents a 2-cell (face).

n-Gmaps with attributes We can add attributes to an n-Gmap, to keep for example the color of the represented object. Attributes can be associated to darts or to cells. For instance if an image is represented by a 2-Gmap and, we want to store the color, it could make sense to associate it to 2-cells (faces).

Additional definitions In the next sections we will use for simplicity the term **gmap** instead of the term generalized map or n-Gmap. We will use the term gmap to refer also to a 2-Gmap since we will use mostly 2-Gmaps. The term **weighted gmap** will be used to refer to a gmap with attributes where a weight has been assigned to each 1-cell (edge). The weight represents an estimate of the geometrical distance between the two incident vertices of each edge. We will use the term **unweighted gmap** or simply **gmap** to refer to a gmap without weights, or equivalently to a gmap with all the weights equal to 1.

1.1.1 Operations

Removal The removal operation consists in removing a given i-cell c while merging the two $(i+1)$-cells incident to c when they exist; when there is only one $(i+1)$-cell incident to c, no cells are merged. Two examples are depicted in Fig. 2. Starting from the 2D object in Fig. 2 (a), first, the two edges e1 and e2 are removed, producing the object given in Fig. 2 (b). The two faces incident to edge e1 in the initial object are merged into one face (this is similar for the two faces incident to edge e2). Then, the two vertices v1 and v2 are removed, producing the object in Fig. 2 (c). The two edges incident to vertex v1 are merged into one edge (as the two edges incident to vertex v2).

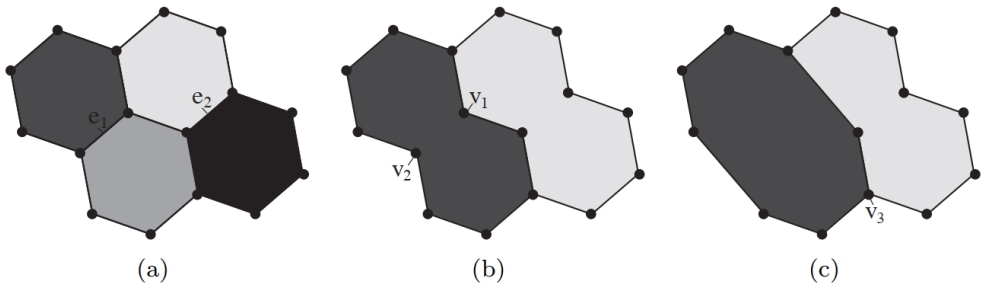

Figure 2: Removal operation examples

Moreover, it is not possible to remove any i-cell: at most two (i + 1)-cells are incident to c, in order to be able to merge these cells when removing c. It is for example impossible to remove vertex v3 in Fig. 2 (c): v3 is incident to three edges, which cannot be merged into one edge.

1.2 Distance transform

Definition The distance transform is a function usually applied to binary images. It computes for each pixel in the foreground the minimum distance from the background.

Propagation metrics Different metrics can be used to propagate the distance. In Fig. 3 the Manhattan distance[1] has been used, so distance can't propagate through diagonals. As showed in the example, distance propagates from the background, represented with zeros, to the foreground, represented with ones.

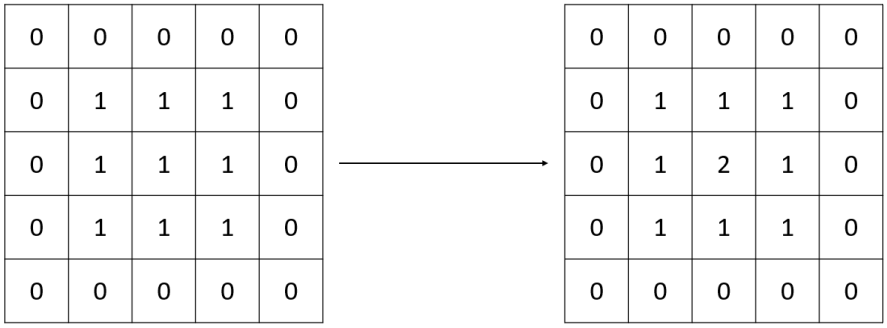

Figure 3: Distance transform example

Generalization of the operator Distance transform can be generalized in order to be applied to non-binary images and to other data structures like graphs and combinatorial maps.

Distance transform of graphs The problem of computing the distance transform of a graph is similar to the **shortest path problem** [6]. The shortest path problem is the problem of computing the shortest path between two vertices in a graph, respectively the source and the destination, where the distance of a path is measured by summing the weights of all the edges that are in the path. For the distance transform problem we have multiple sources and multiple destinations, so we can call the problem **multiple sources and destinations shortest path problem**. We can adapt the existing algorithms that solve the shortest path problem in order to compute the distance transform of a graph. For example for an unweighted graph something similar to the **breadth first search (BFS)** algorithm can be used to compute the distance transform in linear time. In Fig. 4 such algorithm has been applied to a graph.

[1] *Manhattan distance*, `https://xlinux.nist.gov/dads/HTML/manhattanDistance.html`, Accessed: 09-12-2021.

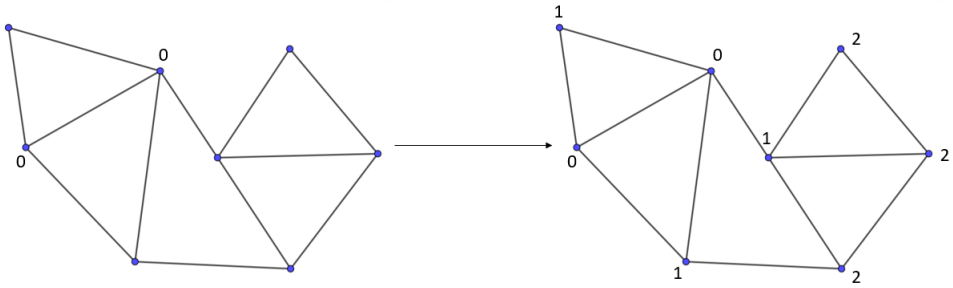

Figure 4: Example of distance transform of a graph

Distance transform of gmaps The problem of computing the distance transform of a gmap is similar to the problem of computing the distance transform of a graph. In fact, we can build a graph from a gmap and then apply an algorithm that solves the problem for a graph. In order to do that we have to decide what are the vertices and the edges and then choose the seeds to use to propagate distance. Multiple choices are possible. The easiest way to do that is to build a graph where a vertex is associated to each dart of the corresponding gmap and an edge exists between two vertices if an involution exists between the two correspondent darts in the gmap. For example in Fig. 5 the dart 0 in the gmap on the left has been associated to the vertex 0 in the graph on the right. Moreover, an edge exists between vertex 0 and vertices 1, 5 and 6 because in the corresponding gmap dart 0 is linked respectively by α_0, α_1 and α_2 to darts 1, 5 and 6. In this way we are using darts as the basic units to propagate distance, and we are propagating distances to all the directions. Other alternatives can be considered, for example, instead of darts, 1-cell (vertices) in the gmap can be used as units as shown in Fig. 6.

After we build the graph an algorithm defined to compute the distance transform of the graph can be applied to obtain the distance value for each vertex.

Actually, we don't need to explicitly build the graph from the gmap. We can define algorithms that directly work for gmaps without explicitly perform the initial transformation to a graph [6]. In Chapter 2 we will present two algorithms that can be used to compute the distance transform of unweighted and weighted gmaps that are based on the idea of the implicit transformation to a graph that we have presented in this chapter.

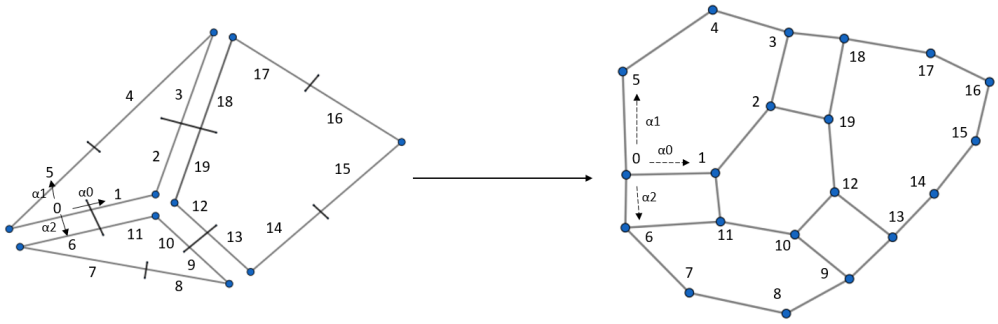

Figure 5: From gmap to graph 1

For simplicity, we will often use in the rest of the document the term **DT** instead of distance transform.

1.3 Water's gateway to heaven project

The Water's gateway to heaven project is a research project bridging **plant biology, 2D-3D imaging, and computer science**. It's a collaboration between three universities of Vienna: plant eco-physiologists and anatomists at the University of Natural Resources and Life Sciences, plant cell biologists at the University of Vienna, and computer scientists expert in pattern recognition and image analysis at the Vienna University of Technology (TU Wien).

The project focuses on the **stomata**, tiny pores on the surface of plant leaves. Stomata play a central role in global water and carbon cycles while covering less than 5% of the leaf surface. Stomata open and close to provide CO2 for photosynthesis and to limit water loss, but this process varies in speed and dynamism between different species and phenotypes, as well as under fluctuating environmental conditions, like under changing light or humidity. The speed at which stomata respond influences the productivity and water use of both crops and natural ecosystems.

The main goal of the project is to answer long-standing questions about **stomatal movements** and to generate basic knowledge on how to **improve**

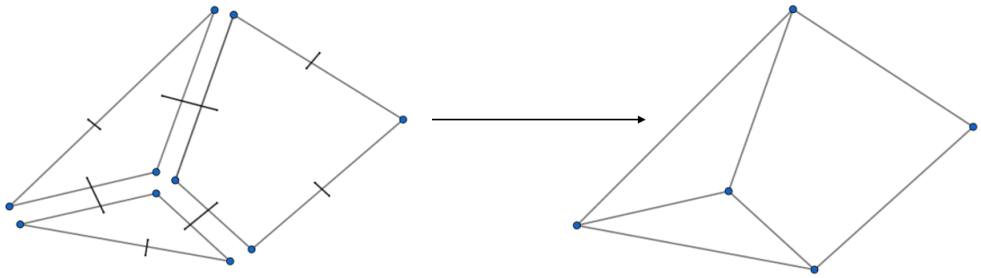

Figure 6: From gmap to graph 2

stomatal responses under dynamic environments in order to increase net productivity and water-use efficiency. In order to do that new computer methods will be used to study 2D and 3D images of leaves obtained combining high-resolution X-ray microtomography and fluorescence microscopy.

More information on the project can be found at[2] and[3].

For simplicity the term **Watergate** will be used instead of Water's gateway to heaven in the rest of the document.

1.3.1 Gas exchange

As described in[4] gas exchange is the physical process by which gases move passively by **diffusion** across a surface.

Flowering plants exchange gases through their leaves through stomata. Carbon dioxide flows from the environment to the inside of the leaf where

[2] *Water's gateway to heaven*, `https://waters-gateway.boku.ac.at/`.

[3] *Water's gateway to heaven prip*, `https://www.prip.tuwien.ac.at/research/current_projects/wgh/`.

[4] Royal society of biology, *Gas exchange*, `https://www.rsb.org.uk/images/12_Gas_exchange.pdf`, Accessed: 01/02/2022.

it's used for photosynthesis while oxygen, obtained from respiration, flows from the inside to the outside. In light condition there is a net intake of carbon dioxide while in dark condition there is a net intake of oxygen.

The rate of gas exchange is affected by multiple factors including:

- the area available for diffusion
- the concentration gradient across the gas exchange surface
- the speed with which molecules diffuse through membranes
- the distance over which diffusion occurs

Efficient gas exchange systems must:

- have a large surface area to volume ratio
- be thin
- have mechanisms for maintaining steep concentration gradients across themselves
- be permeable to gases.

1.4 Goals and motivations

This work has two goals:

- to design and implement algorithms to compute the distance transform of gmaps (unweighted and weighted).
- to apply such algorithms for the Watergate project to study the gas exchange process.

Design and implement algorithms to compute DT of gmaps As described in Section 1.2 the DT operator is defined for binary images but can be also defined for other data structures[5], [6], [10]. Since we have not found algorithms in the literature to compute the distance transform operator other than the one proposed in [6], the first goal is to design and implement such algorithms to make them available to the scientific world. In particular, we want to implement two algorithms, one for unweighted gmaps and one for weighted ones. Moreover, the algorithms should be enough general to be applied to an n-dimensional gmap of an arbitrary n, and they should work with or without obstacles. Finally, the algorithms should be enough efficient to be applied to large generalized maps.

Study the gas exchange process In order to achieve the main goal of the Watergate project described in Section 1.3 it's important to study the gas exchange process that occurs in plant leaves. Since the **gas exchange rate** depends on the distance over which the diffusion occurs and on the area available for diffusion[6], we want to apply the algorithms that have to be defined as first goal to compute such parameters. In fact the distance transform operator can be used to compute the geodesic distance from stomata to cells (where CO2 is used and O2 is produced) and to obtain Voronoi diagrams that can be used to study the exchange area for each stomata. The DT will be applied to gmaps that have been used to keep images of the leaves. A gmap can be reduced applying the operation described in Section 1.1. This is crucial since the size of the available data is huge and using the gmaps we can reduce the size to keep only the sufficient amount of information needed. For the computation of the diffusion distance we will analyze the trade-off between speed and accuracy with different reduction factors.

In the following chapters we will use the term **diffusion distance** to refer to the distance from stomata to cells.

[5] *Distance transform*, `https://homepages.inf.ed.ac.uk/rbf/HIPR2/distance.htm`, Accessed: 02/02/2022.

[6] Royal society of biology, *Gas exchange*, `https://www.rsb.org.uk/images/12_Gas_exchange.pdf`, Accessed: 01/02/2022.

2 Proposed solutions

In this section we will present all the algorithms. The implementation is in Python (3.9) and all the code is available in a GitHub repository[7].

2.1 Gmap implementation

We need an implementation of the gmap to define and test the algorithms.

The PRIP lab has provided an implementation in Python based on the one described in [2]. It consists in a hierarchy of classes which have their root in the **nGmap** class, that is an array based implementation of the gmap, where a set of arrays is used to store the involutions (a dict based implementation has been provided but for our application the array based is better). The **PixelMap** class extends nGmap and specifically represents a 2-Gmap. **LabelMap** extends PixelMap and adds the possibility to build a 2-Gmap from a list of labels. For this work both PixelMap and LabelMap are used.

The provided implementation has been modified in multiple parts. In particular, we added attributes to keep additional information, like weights, in order to distinguish between unweighted and weighted gmaps.

Moreover, the memory consumption has been reduced by a factor of 16.62, allowing to perform experiments on big images even on a machine with a limited amount of memory. In fact before the update a gmap built from a 1000x1000 image would have occupied 5.33 GB, while after the update it occupies 328.13 MB.

2.2 DT algorithm for unweighted gmaps

2.2.1 Naive algorithm

A naive algorithm can be derived from [6] and by adapting the breadth-first search **(BFS)** algorithm to gmaps. The easiest way to compute the DT of a gmap is to use darts as units and propagate distance to every direction, i.e. to every neighbor that can be reached applying all the involutions to the current dart.

[7] C. Carratù, *Distance transforms on gmaps*, `https://github.com/LordCatello/distance-transforms-on-gmaps`.

More formally given a gmap $G = (D, \alpha_0, ..., \alpha_1)$ and a set of seed darts $S \subset D$ the algorithm propagates the distance starting from darts in S considering as neighbors of a dart d:

$$N(d) = \{\alpha_i(d) \;\; \forall i \; from \; 0 \; to \; n\} \tag{2}$$

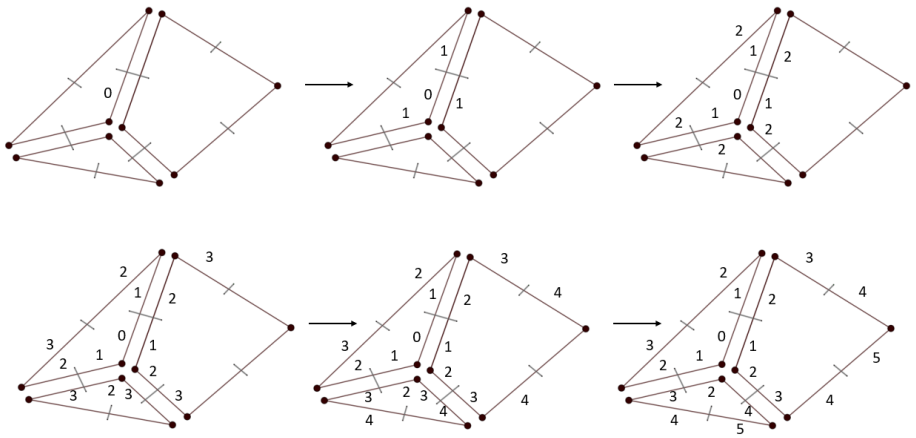

Figure 7: Example of how naive algorithm works

In Fig. 7 the algorithm has been used to compute the DT of the depicted gmap. Only one seed has been chosen, and its distance has been initialized to 0. At each step distance propagates (increasing by 1 unit) from each non-visited dart to its neighbors. In the example at the first step distance propagates from the seed s to its neighbors $\alpha_0(s)$, $\alpha_1(s)$ and $\alpha_2(s)$. After 5 steps distance has been computed for all the darts.

Pseudocode We can analyze the pseudocode (python like) depicted by Algorithm 1 to better understand how the algorithm works.

The function receives a gmap and a list of seeds as input and computes the DT for each dart of the gmap. The distance value is saved in an array where the indices identify the darts. The array is defined as an attribute of the gmap class. The algorithm includes an initialization phase, from Line 2 to 10. First distances are initialized to -1. That value is used to mark a dart as not visited. Then a **queue** is instantiated and all the seeds are added

Algorithm 1 Naive algorithm

Input: *gmap*, *seeds*

```
1:
2: for dart in gmap.darts do                          ▷ Initialize distances to -1
3:     gmap.distances[dart] = -1
4: end for
5:
6: queue = Queue()
7: for seed in seeds do                                ▷ Add seeds to queue
8:     gmap.distances[seed] = 0
9:     queue.put(seed)
10: end for
11:
12: while not queue.empty() do
13:     dart = queue.get()
14:     for i in range(gmap.n + 1) do                  ▷ Visit all the neighbours
15:         neighbour = gmap.ai(i, dart)
16:         if gmap.distances[neighbour] == -1 then    ▷ If the neighbour has
    not been visited yet
17:             gmap.distances[neighbour] = gmap.distances[dart] + 1
18:             queue.push(neighbour)
19:         end if
20:     end for
21: end while
```

to it. Moreover, the distance of each seed is set to 0. From Line 12 starts the main loop. At each iteration a dart is removed from the queue. All the neighbors of the dart are visited and each of them, if not previously visited, is added to queue, and its distance is set to the distance of the current dart plus 1.

At the end of the main loop, at Line 21, all the darts have been visited, and the DT has been computed for all of them. Since the algorithm is an adaptation of BFS, darts are visited by level, increasing distance at each level.

Complexity Worst case complexities are:

$$Time = O((n+2)|D|) \tag{3}$$

$$Space = O(D) \tag{4}$$

where $|D|$ is the number of darts and n is the dimension of the gmap (1-Gmap, 2-Gmap, ...).

2.2.2 Generalized algorithm

The Naive algorithm is very limited since **only darts can be used as units to propagate distance** and, **it can't be used to compute geodesic distance** since it's not possible to define an obstacle. We want to solve both problems implementing a more generalized algorithm.

Use a generic i-cell to propagate distance For the first problem we want to use an arbitrary cell as unit to propagate distance, e.g. we want to use vertices or edges.

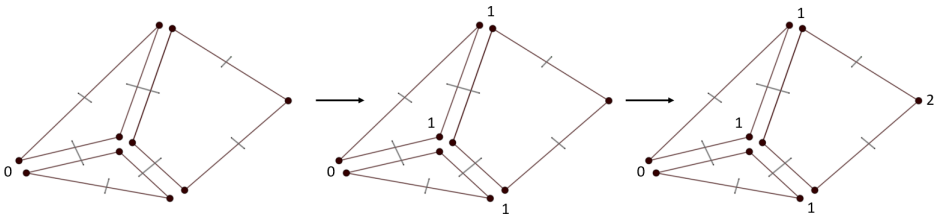

Figure 8: Distance is propagated using vertices as units

The Fig. 8 shows how the algorithm should work if vertices are used to propagate distance. To solve the problem we decided to keep the implementation of the Naive algorithm that associates distance to each dart, but we have defined a mechanism whereby **distance does not increase for some darts**. In fact, since we are associating distance to darts, we want to achieve that **all darts of the same i-cell chosen as unit, must have the same distance**, so distance must increase only from darts of one i-cell to darts of another i-cell.

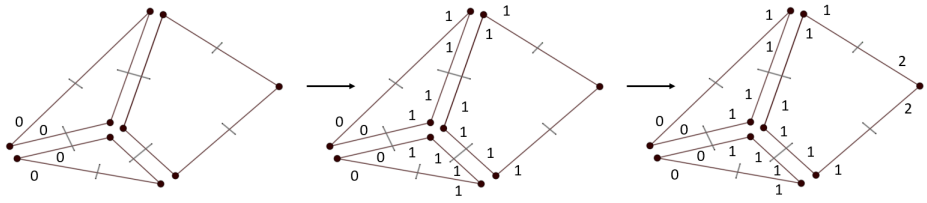

Figure 9: All darts of the same vertex share the same distance value

In Fig. 9 is depicted how the algorithm should work: darts of the same vertex must have the same distance. That behavior has been achieved increasing the distance only if the current dart has been reached by a specific involution. In particular, distance must increase only through α_i if we want to use i_cell has units. If we want to use vertices (0-cell) as units, we must increase distance only if a dart is reached through α_0. More precisely **distance should be the same in the orbit of each the i-cell**, where the orbit of an i-cell is the set of all the darts of that cell. Fig. 10 shows that in order to move from a vertex v to another it should be computed involution α_0 of darts inside v.

So we added the input **accumulation_directions** to the algorithm. It's an array of boolean, where each element is True for the index i if distance should increase for the involution i. In a 2-Gmap to use vertices as units the array should be equal to [True, False, False] while to use edges it should be equal to [False, True, False].

Manage obstacles To manage obstacles and compute geodesic distance we have to define an obstacle. The simplest way to do that is to associate a label to each dart and propagate distance only if the label of the current dart is in a list of admitted labels. All darts with a different label are considered

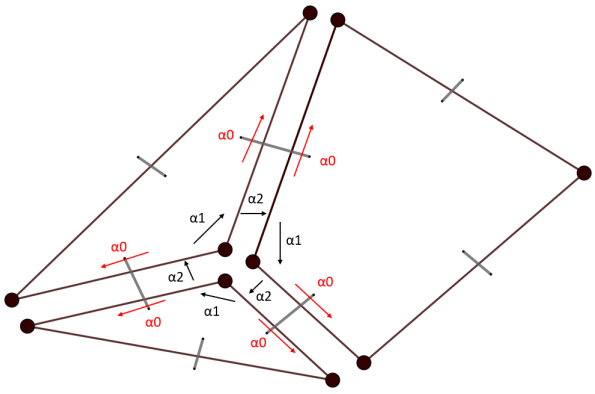

Figure 10: The orbit of a vertex can be obtained applying α_2 and α_1 to a random dart of the vertex

as obstacles and not visited. So we add a parameter to the algorithm whose value is the **set of permitted labels**.

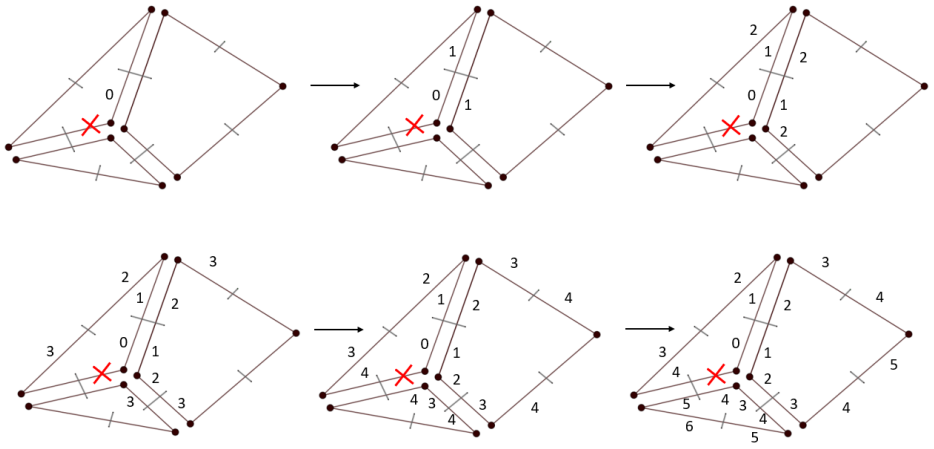

Figure 11: DT example with propagation labels

In Fig. 11 we applied the DT algorithm to the same gmap with the same initialization of the example in Fig. 7 but marking as prohibited the dart with the red cross. Due to the fact that distance cannot propagate to that dart, we obtain that it propagates slowly to the bottom face.

Complexity The generalized algorithm has the same worst case time and space complexities as the Naive one.

2.3 DT algorithm for weighted gmaps

We need an algorithm to compute the DT of weighted gmaps. The algorithm should be generalized as the one for unweighted gmaps. To do that we can adopt the very same approach used for unweighted gmaps, i.e. we can employ accumulation directions to use a generic i-cell and a set of permitted labels to manage obstacles. Since we have already discussed the generalization in Section 2.2.2 we will now describe only the differences between the two approaches.

The algorithm is an adaptation of Dijkstra's algorithm for graphs. We decided to associate weights to i-cells. In particular, for the application to study gas exchange, we associated weights to edges (1-cell). To do that a weight has been associated to each dart but every dart of the same edge share the same weight, as depicted in Fig. 12

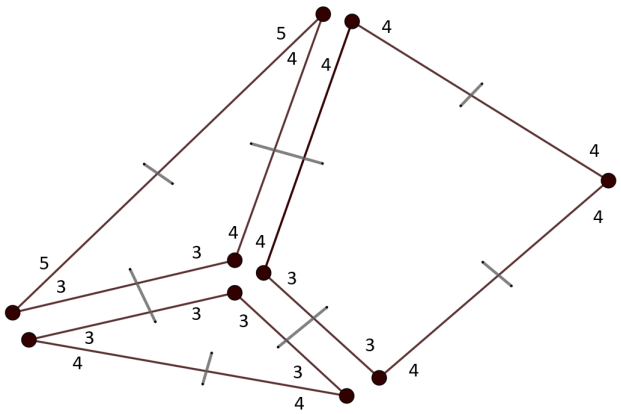

Figure 12: Darts in the orbit of an edge share the same weight

Pseudocode The naive function (which does not manage obstacles and, it's not generalized for cells) described by Algorithm 2 receives a gmap and a list of seeds as input. In the initialization phase, from Line 2 to 12 distances are initialized to -1 and seeds are pushed to a **heap**. Moreover, a **set** is instantiated. It will keep all darts already visited. From Line 14 starts the

Algorithm 2 Naive algorithm for weighted gmaps

Input: *gmap*, *seeds*

```
1:
2: for dart in gmap.darts do                          ▷ Initialize distances to -1
3:     gmap.distances[dart] = -1
4: end for
5:
6: heap = Heap()
7: for seed in seeds do                                ▷ Add seeds to heap
8:     gmap.distances[seed] = 0
9:     heap.push(seed, gmap.distances[seed])
10: end for
11:
12: visited = Set() ▷ visited is used to keep track of all darts removed from
    the heap
13:
14: while not heap.empty() do
15:     dart, _ = heap.pop()                  ▷ Pop dart with minimum distance
16:     visited.add(dart)
17:     for i in range(gmap.n + 1) do                  ▷ Visit all the neighbours
18:         neighbour = gmap.ai(i, dart)
19:         if neighbour not in visited then
20:             if gmap.distances[neighbour == -1] or gmap.distances[dart] +
    gmap.weights[dart] < gmap.distances[neighbour] then
21:                 gmap.distances[neighbour] = gmap.distances[dart] +
    gmap.weights[dart]
22:                 heap.update(neighbour, gmap.distances[neighbour])      ▷
    Update entry with key neighbour if exists otherwise add a new entry
23:             end if
24:         end if
25:     end for
26: end while
```

main loop. At each iteration **the dart with the smallest distance is removed from the heap**. Then the weight of each non visited neighbour is updated and added to the heap (Line 21).

At the end of the main loop, at Line 26, all darts have been visited.

Complexity Worst case complexities are:

$$Time = O((n+1)|D| + |D| \log D) \tag{5}$$

$$Space = O(D) \tag{6}$$

where $|D|$ is the number of darts and n is the dimension of the gmap (1-Gmap, 2-Gmap, ...).

2.4 Algorithm to compute diffusion distance and Voronoi diagrams

As mentioned in Section 1.4 we want to use the algorithms defined in Section 2 to study the gas exchange process. In particular, we are interested in computing the diffusion distance, that is the distance from stomata to cells, and Voronoi diagrams that can be used to study the exchange are for each stomata.

More precisely, we have in input a labeled 2D segmented image of a section of a leaf as depicted in Fig. 13, and we want to compute:

- The **diffusion distance**, that is the average distance computed from the borders of stomata (in light grey in the example) to the borders of cells (in black in the example)
- A **Voronoi diagram** where a different propagation region has to be identified for each stomata.

A pipeline has been implemented in order to achieve both goals. It is described in the next subsections.

2.4.1 Reduce image size

In order to **speed up the process** the size of the image can be first reduced using a median filter as depicted in Fig. 14. This is an optional step that has been performed to speed up the experimentation phase. The **median filter** works by choosing for each pixel in the output image the most common pixel under the kernel in the input image.

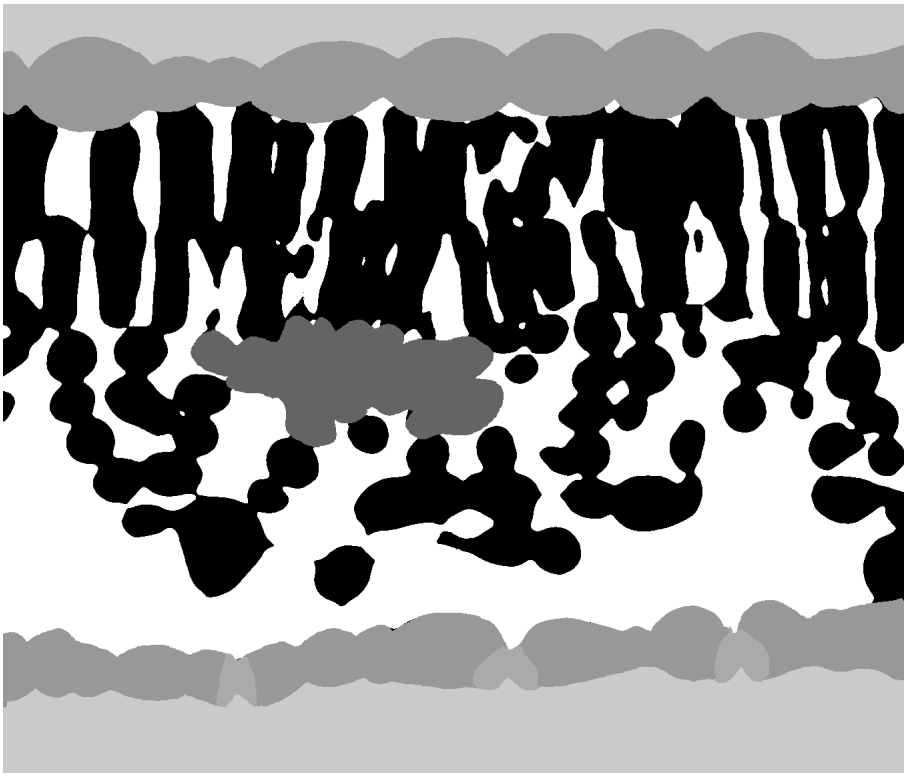

Figure 13: Example of a labeled 2D segmented image of a cross-section of a leaf

2.4.2 Label connected components

To generate Voronoi diagrams we need each stoma to be uniquely identified by a distinct label, so that we can construct a distinct region for each of them. Unfortunately, the labeled input images used in the experimental phase have only 6 labels, identifying different parts of the cell, such as air or veins, as described in Section 3.1. Since in these images all the stomata share the same label, we decided to apply an algorithm to **label the connected components**, precisely the one pass algorithm, to assign a unique label for each stoma. In Fig. 15 is depicted an example where the algorithm has been applied to an image. The output is an image of the same size of the original one where the value of each pixel is an integer that identifies the connected component to which the pixel belongs. It can be observed that now each stoma is represented with a distinct color (randomly chosen), since each of

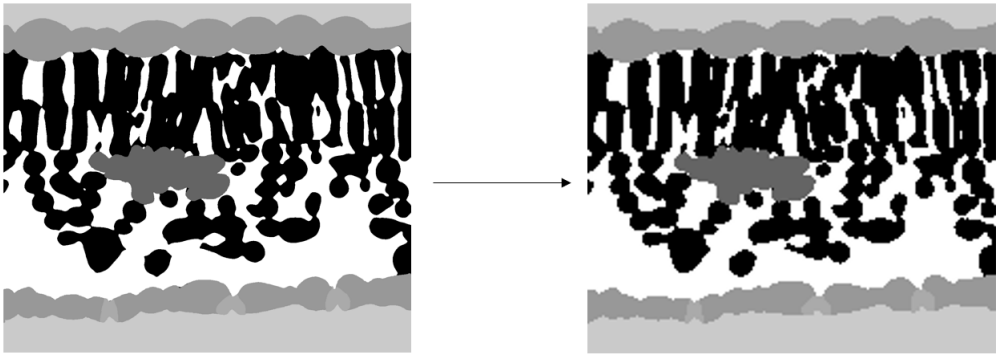

Figure 14: Example of an original (1115x1350) and reduced (165x193) image using a 7x7 median filter

them has a unique label.

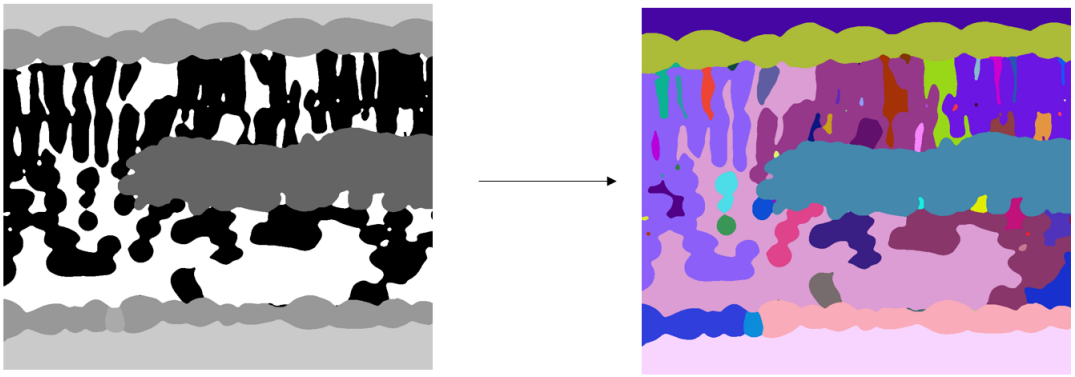

Figure 15: The image on the right is obtained by applying the one pass algorithm to the image on the left. Each color identifies a distinct connected component

2.4.3 Build and reduce the gmap

After having labeled the connected components a 2-Gmap must be build from the reduced image as depicted in Fig. 16. The labels obtained in the previous step are saved as an attribute.

A gmap can be reduced applying the operation described in Section 1.1, merging some regions with the same label as depicted in Fig. 17. This could

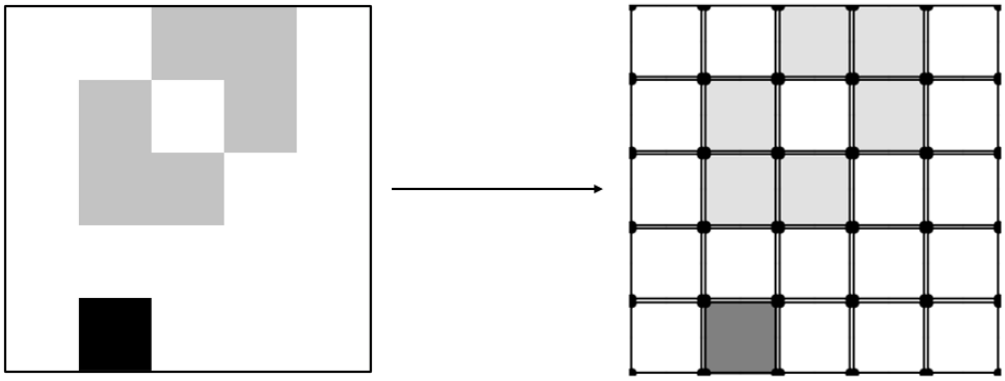

Figure 16: The gmap on the right is obtained from the image on the left

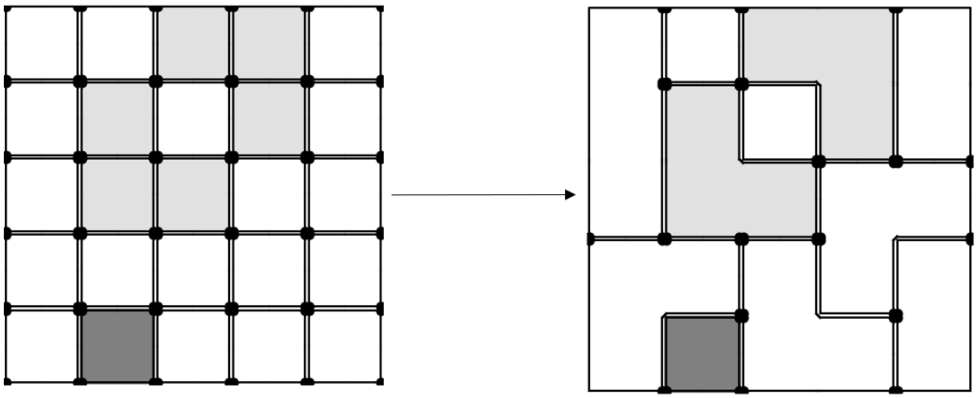

Figure 17: A gmap can be reduced applying a set of removal operations

be useful since the computation of the distance transform will require less time on a reduced gmap, so the goal of this step is to improve time performance. On the other hand the computation will be less accurate, since we are losing information merging regions with the same label, so we decided to introduce a parameter, the (**reduction factor**), that can be used to specify the number of edges to remove among those that can be removed. The parameter can be used to adjust the trade-off between speed and accuracy that we have reducing the gmap, and we will analyze in Section 3 how performance and accuracy change with different values of the reduction factors.

Weights update During the reduction process weights have to be updated as depicted in Fig. 18. Weights are associated to darts and each dart of the same edge must have the same weight. Weights are updated when a vertex removal occurs. The weight of the resulting edge must be equal to the weights of the merging edges. The new weight can be computed as:

$$weight(d) + weight(\alpha_1(d)) \tag{7}$$

where d is one of the darts to remove.

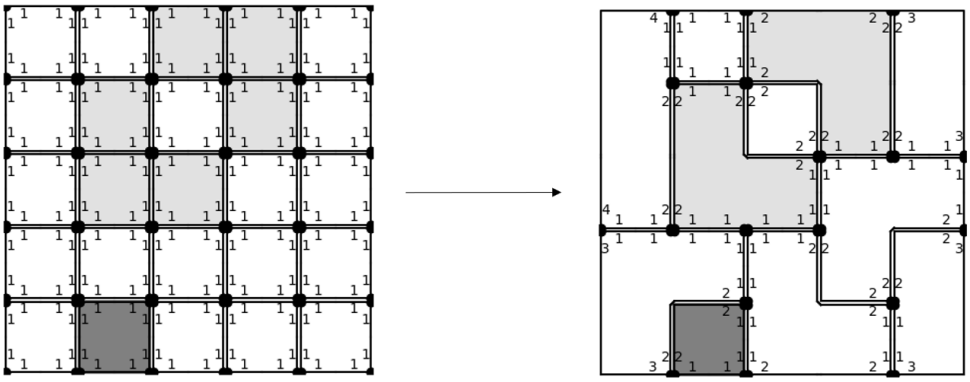

Figure 18: Example of weights update after some removal operations

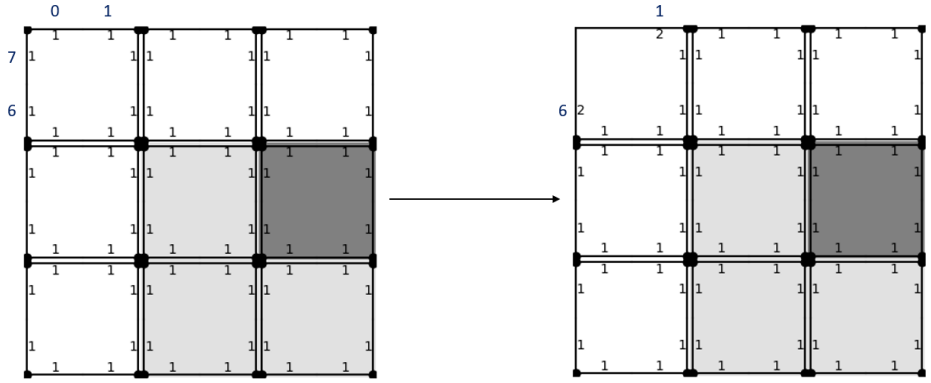

Figure 19: Example of weights update after removing the vertex identified by darts 0 and 7

In the example depicted in Fig. 19, where dart identifiers are coloured in blue and weights in black, the vertex identified by darts 0 and 7 has been removed. The weight of the resulting edge, identified by darts 1 and 6 has to be updated to $weight(0) + weight(\alpha_1(0))$ or equivalently to $weight(7) + weight(\alpha_1(7))$, so it will be equal to 2. As you can see the value of the weight of an edge is shared among the darts in the orbit of that edge, so for example, in this particular case, darts 1 and 6 share the same weight that is equal to 2.

2.4.4 Compute DT

The DT has to be computed from stomata to the border of cells, using intracellular space (in white in the labeled images) to propagate. A **contour plot** (Fig. 20) can be generated to better analyze the result. Darker colors are used for shorter smaller, while lighter colors are used for greater distances.

The DT can be computed using the algorithm for unweighted gmaps or the one for weighted ones. If the gmap is reduced it would seem to make more sense to use the algorithm that takes into account the weights since some edges will have lengths greater than 1. Experiments show that this is not always the case, so sometimes it's better to use the algorithm for unweighted gmaps even if the gmap is reduced. This is explored in detail in the experiments section.

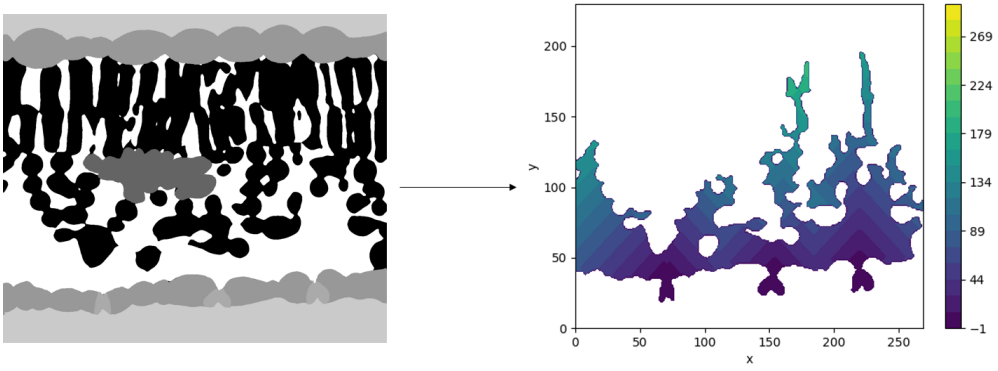

Figure 20: A contour plot can be generated to better analyze the result

2.4.5 Compute diffusion distance

The diffusion distance is computed by taking the average of the distances for borders of the cells. The value is multiplied by the image reduction factor if the image has been previously reduced at the step described in Section 2.4.1

2.4.6 Generate Voronoi diagrams

To generate Voronoi diagrams we need to know for each dart from which stomata it has been reached during the computation of the DT. So we need to modify the algorithms used in Section 2.4.4 to save this information. Given the label of stomata saved for each dart an image as the one depicted in Fig. 21 is generated.

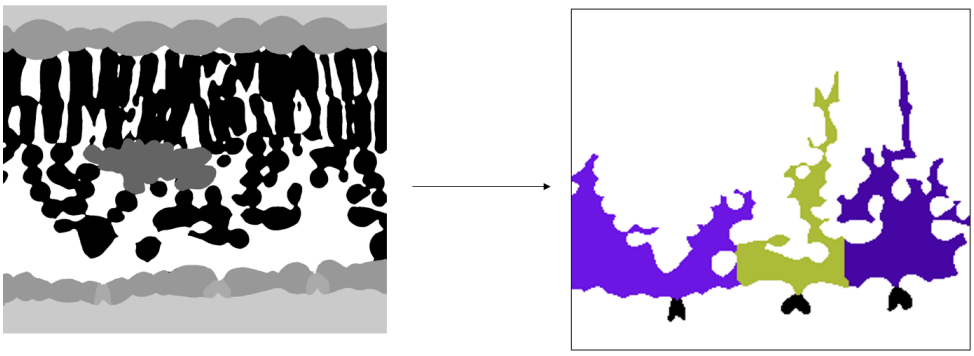

Figure 21: Each region of the Voronoi diagram identifies the propagation area of the correspondent stomata

These images can be used to understand which stomata contributes the most to the propagation of gas.

3 Experiments

Experiments have been performed to compute the diffusion distance and generate Voronoi diagrams to provide information to biologists and to analyze the trade-off between accuracy and speed when reducing the gmap.

3.1 Dataset

The dataset consists of 2D images of a leaf collected in July - October 2020. The images are slices of 3D data obtained by **high-resolution X-ray micro-tomography (μCT)**. There are in total 209 labeled images of 5 time steps of 3 kinds:

- **paradermal** 1350x1119 slices
- **cross-sect (cross)** 1450x1152 slices
- **long-sect (long)** 1152x1119 slices

There are 6 different labels, identified by an uint8 value: cells (0), veins (100), epidermis (152), stomata (170), background (202), air (255). The images were labeled manually by experts. Segmentation algorithms are being developed by other members of the team working on the project to speed up the labeling process. In Figs. 22, 23, 24 are shown examples of, respectively, paradermal, cross and long slices. In each figure, the image on the left is the raw image, while the image on the right is the labeled one.

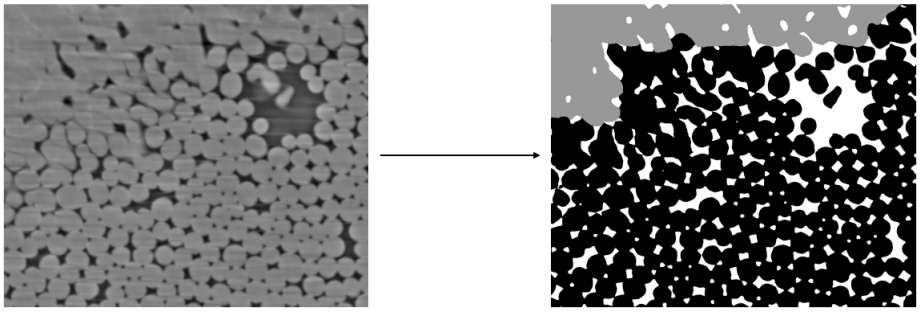

Figure 22: Paradermal slice

For our problem only cross and long slices can be used, since we need vertical slices to compute the diffusion distance, while paradermal slices cut

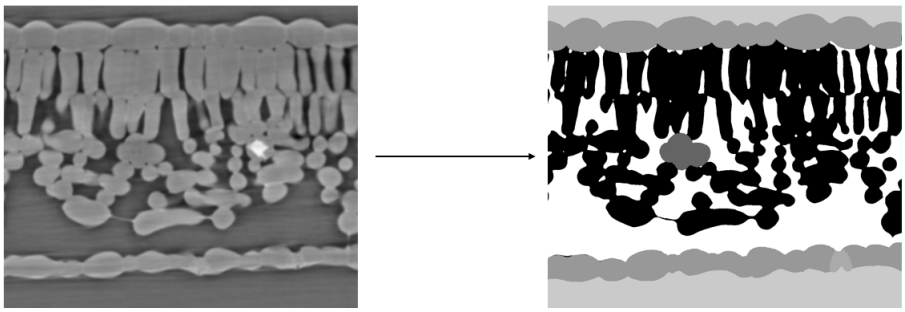

Figure 23: Cross slice

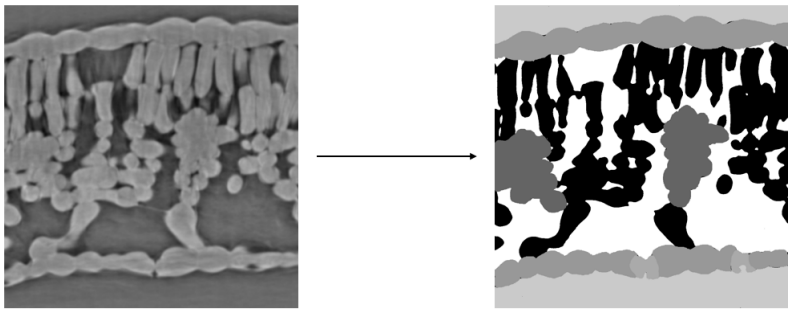

Figure 24: Long slice

the leaf horizontally. So we can use 140 images. Of those, only images with stomata are effectively used (61 images).

3.2 Metrics

Results produced for each image have been combined to obtain an aggregate report. The metrics in the report are:

- **Gmap reduction factor - RF**: the reduction factor used to reduce the gmap. It is a float ranging from 0 to 1. If the value is 0 no reduction is performed, otherwise when the value is 1 all the removable edges are removed;
- **Number of darts - ND**: average number of survived darts after reducing the AMAP. The speed of the algorithms depends directly on the number of darts;

- **Use weights - UW**: whether to use weight to compute the distance transform. If False, the algorithm for unweighted gmaps is used, otherwise it will be used the algorithm for weighted ones;

- **Diffusion distance - DD**: the average of the diffusion distance among all experiments;

- **Diffusion distance error - DDE**: the average of the diffusion distance relative error. The value is equal to $|\frac{base-current}{base}|/n$ where *base* is the diffusion distance value computed without reducing the gmap, *current* is the actual value and n is the number of samples. The error is presented in the form $E \pm 2SD$ where SD is the standard deviation;

- **Time improvement factor - TIF**: the average among all samples of the time used to compute DT without removals (Gmap reduction factor of 0) divided by the time used with the current reduction factor $(\frac{base}{current}/n)$.

3.3 Results

As written in the previous section results produced for each image have been combined to obtain an **aggregate report**. Since experiments have been performed separately for cross and long slices two reports have been generated. The images have been reduced with an image **reduction factor of 11** (see Section 2.4.1 for more details) before building the gmap to speed up the computation.

RF	ND	UW	DD	DDE	TIF
0	92496	False	394.93	0.00 ± 0.00	1.00
0.25	62584	False	352.19	0.10 ± 0.08	1.48
0.25	62584	True	414.86	0.05 ± 0.06	1.94
0.5	31566	False	347.84	0.12 ± 0.12	2.81
0.5	31566	True	505.76	0.29 ± 0.26	3.63
1	730	False	44.25	0.88 ± 0.08	123.98
1	730	True	1121.51	1.75 ± 1.90	172.18

Table 1: Aggregate report for cross slices

RF	ND	UW	DD	DDE	TIF
0	76704	False	403.64	0.00 ± 0.00	1.00
0.25	52276	False	355.56	0.11 ± 0.08	1.46
0.25	52276	True	418.73	0.04 ± 0.06	1.93
0.5	26260	False	337.45	0.16 ± 0.16	2.81
0.5	26260	True	482.54	0.21 ± 0.22	3.67
1	518	False	34.84	0.91 ± 0.08	130.67
1	518	True	921.35	1.25 ± 1.54	198.38

Table 2: Aggregate report for long slices

3.3.1 Quantitative analysis

Experiments have been performed with different gmap reduction factors. Each row of Tables 1 and 2 contains the results, averaged over the number of images analyzed, obtained by reducing the gmap by the reduction factor indicated in RF. For each value of the RF experiments have been performed twice, using or without using weights. When the RF is equal to zero it's not necessary to repeat the experiment since all weights are equal to 1 and so there is no difference in using them. Results obtained with the RF equal to zero are used as baseline to compute the error and the time improvement factor. **Results obtained for cross and long images are similar**, so we will not distinguish in this analysis.

Analyzing the results, we can see that the error tends to increase as the number of darts decreases, while the time taken to compute the DT decreases. Moreover, **the value of the standard deviation is very high**. This means that the error can vary considerably depending on the sample.

Use weights We observe that when weights are not used the diffusion distance value is lower than the baseline. This is understandable since when weights are not used the length of the edges is underestimated as it is always equal to 1. On the contrary when weights are used we could expect more precise results because the algorithm takes into account the length of the arc. Instead, the diffusion distance value is overestimated, and the error increases reducing the number of darts. The result is explained by the fact that **reducing the number of darts lengthens the distance between vertices in the gmap**, because the number of edges is reduced, as depicted

in Fig. 25.

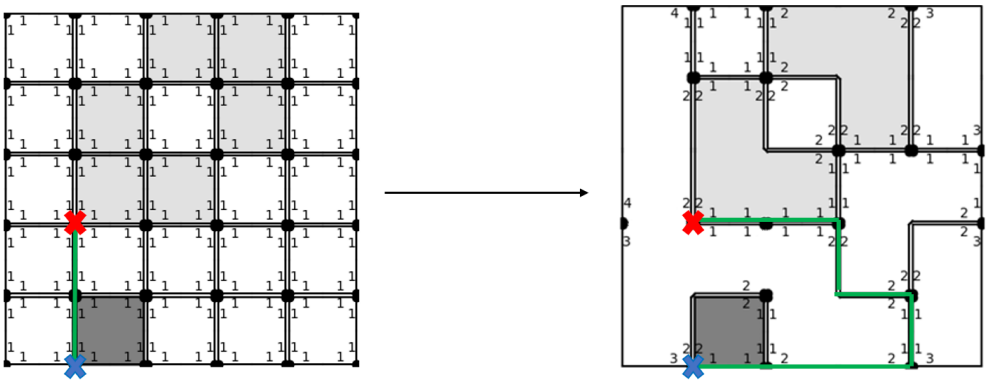

Figure 25: It could happen that the distance between two points increases when reducing the gmap

It seems that up to the RF of 0.25 the weighted algorithm is more accurate, then the error grows faster than the unweighted one. The weighted algorithm is also faster, but this may be due to a more efficient implementation than the unweighted algorithm.

In conclusion, it can be seen that up to a value of 0.5 for the reduction factor, acceptable estimates of the diffusion distance can be obtained. When the value is 1 we obtain a huge speed boost, but the error increases drastically. Further experimentation could be performed with reduction factor values between 0.5 and 1.

3.3.2 Qualitative analysis

In this section we will analyze contour plots and Voronoi diagrams obtained for two samples (one cross and one long).

Cross sample The labeled cross slice is depicted in Fig. 26. In Figs. 27 and 28 are depicted respectively the contour plot and the Voronoi diagram obtained with RF equal to 0. These images are used as reference to compare the results obtained with RF greater than 0.

Speaking about contour plots, images confirm that when using weight the DT is overestimated, while when not using weights it's underestimated, as depicted for example in Fig. 30. When RF is equal to 1 there are too few

darts to obtain an acceptable estimate as can be seen in Fig. 31: the DT is completely different from the one obtained without reducing the gmap at all. Speaking about Voronoi diagrams, there are no difference is using or not using weights. This is due to the fact that in this image the propagation areas of the two stomata are not adjacent. When RF is equal to 1, similarly to what happens for the contour plots, Voronoi diagrams are very unreliable.

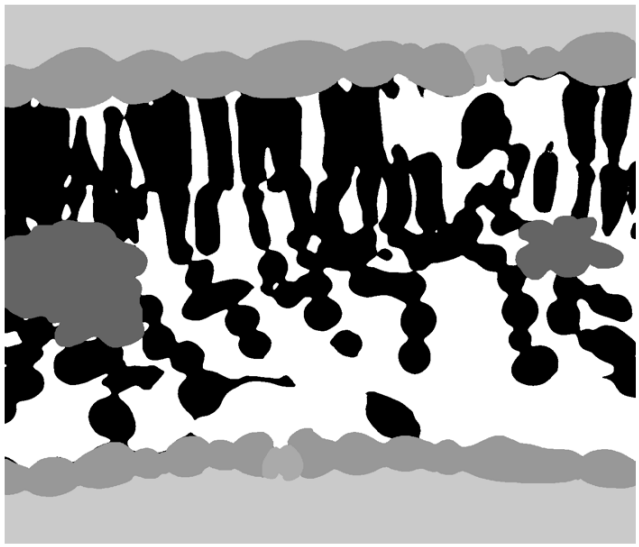

Figure 26: Labeled cross slice

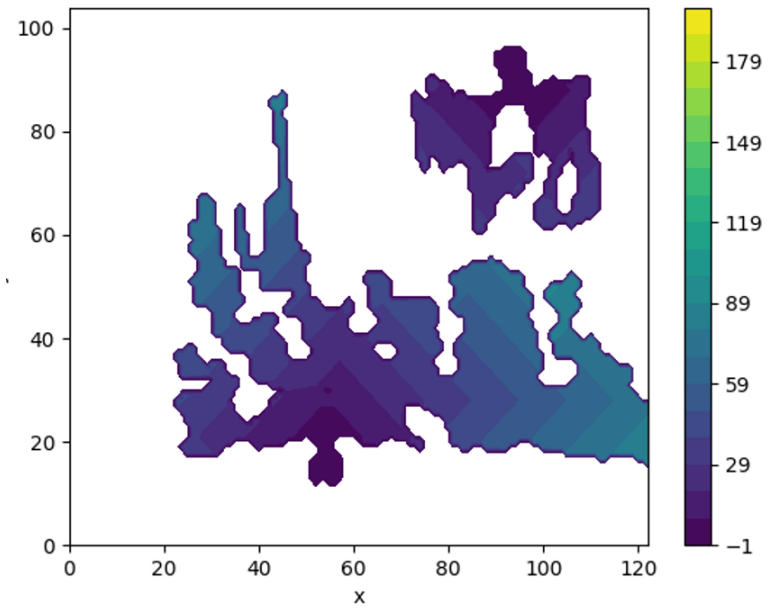

Figure 27: Contour plot with RF equal to 0 of cross slice

Figure 28: Voronoi diagram with RF equal to 0 of cross slice

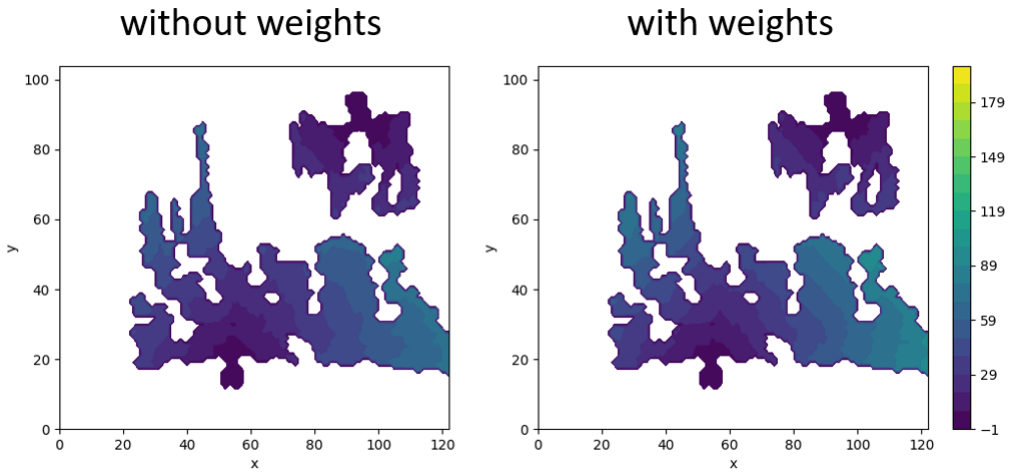

Figure 29: Contour plot with RF equal to 0.25 of cross slice

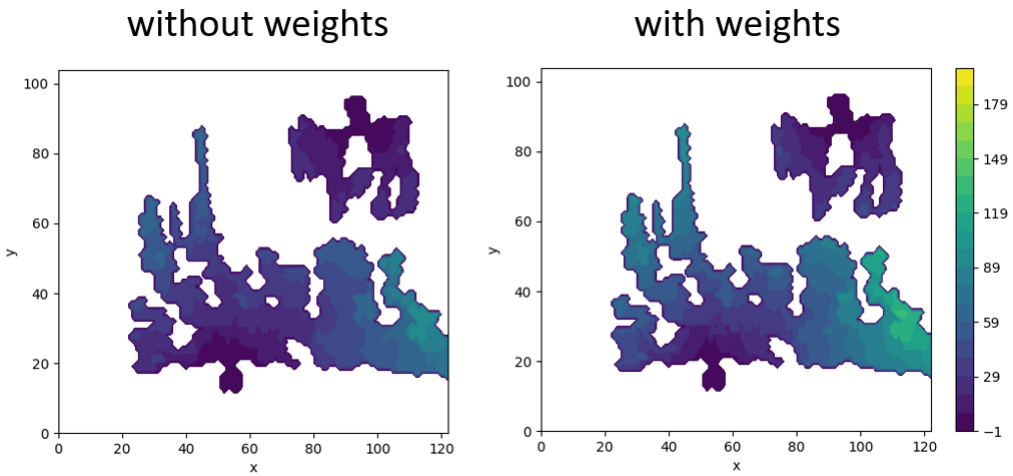

Figure 30: Contour plot with RF equal to 0.5 of cross slice

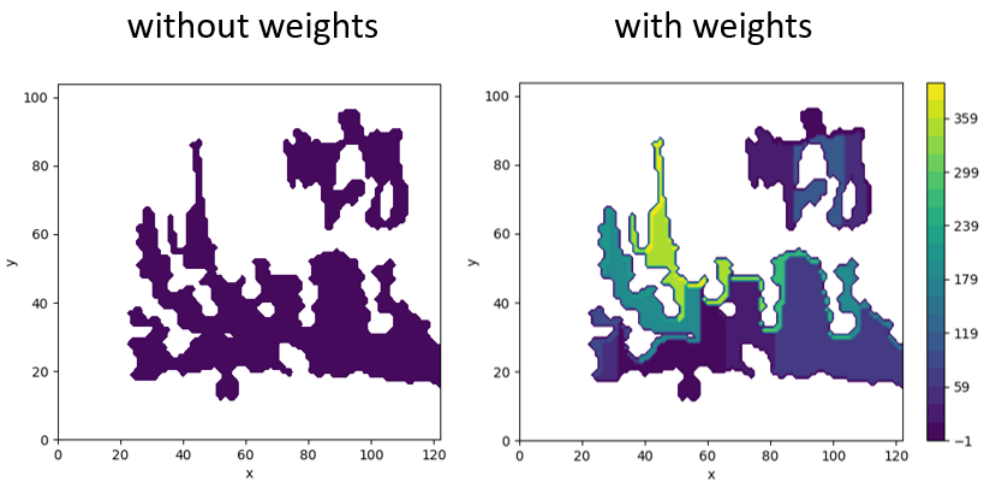

Figure 31: Contour plot with RF equal to 1 of cross slice

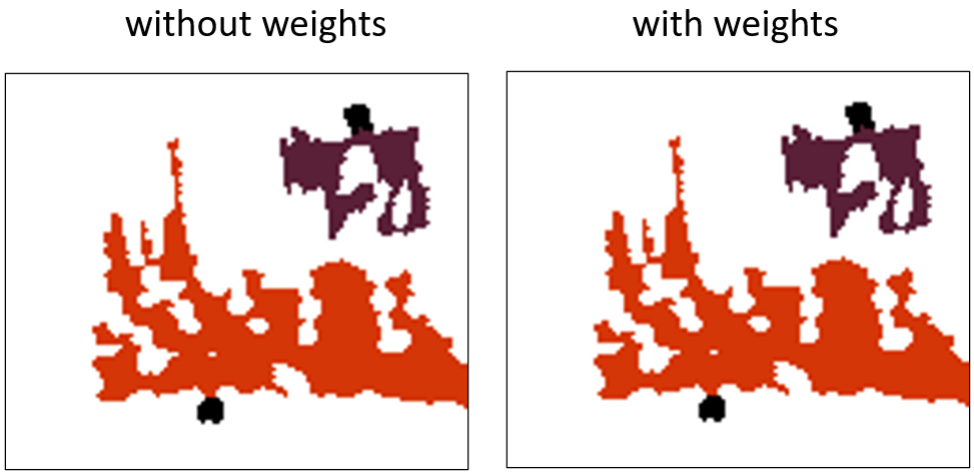

Figure 32: Voronoi diagram with RF equal to 0.25 of cross slice

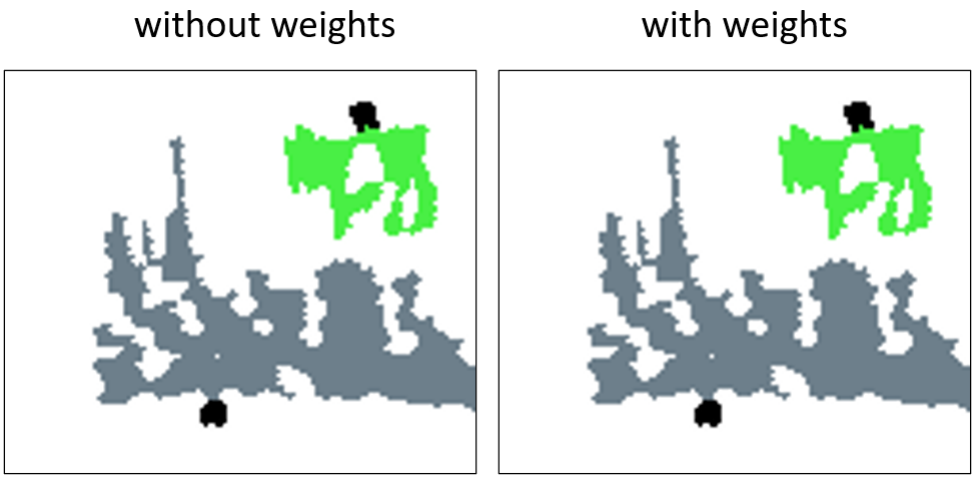

Figure 33: Voronoi diagram with RF equal to 0.5 of cross slice

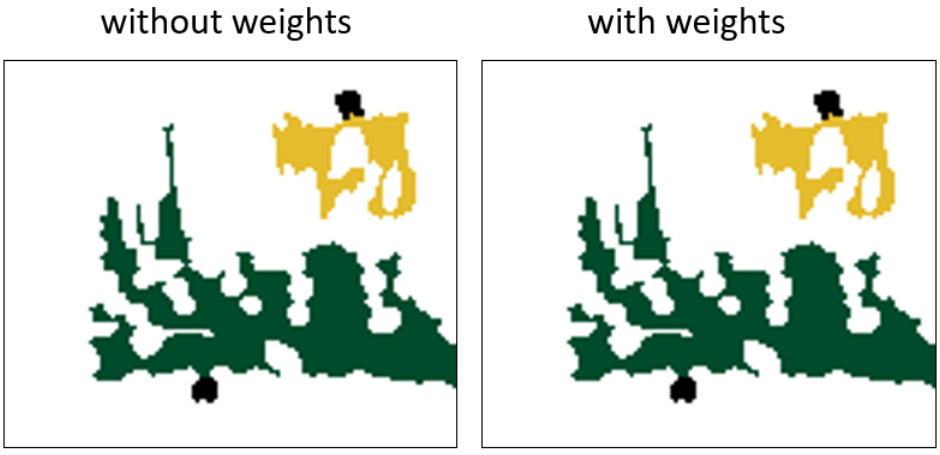

Figure 34: Voronoi diagram with RF equal to 1 of cross slice

Long sample The labeled long slice is depicted in Fig. 35. In Fig. 36 and 37 are depicted respectively the contour plot and the Voronoi diagram obtained with RF equal to 0 and are used as reference.

Regarding the long slice, similar observations can be made to those already made for the cross slice. However, in this case the propagation areas of the stomata are adjacent and the Voronoi diagram obtained using weights is different from the one obtained without using them. The difference is not relevant up to RF equal to 0.5 (the ownership of a small portion of the propagation area changes between the two stomata if weights are used). The difference is instead observable with RF equal to 1.

Figure 35: Labeled long slice

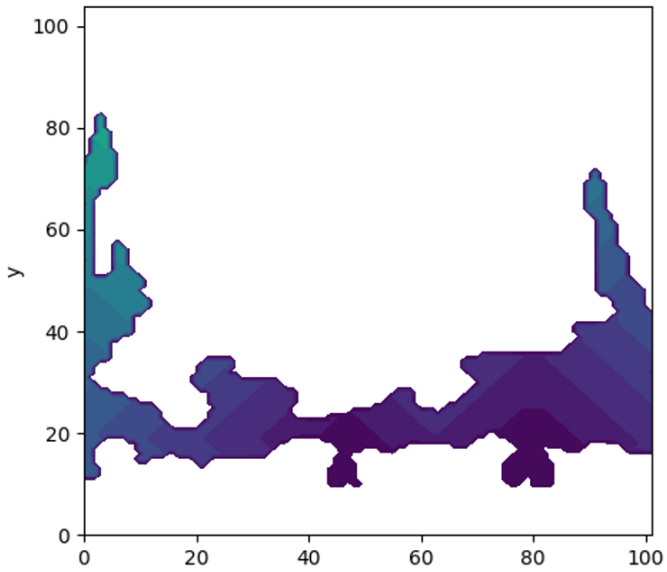

Figure 36: Contour plot with RF equal to 0 of long slice

Figure 37: Voronoi diagram with RF equal to 0 of long slice

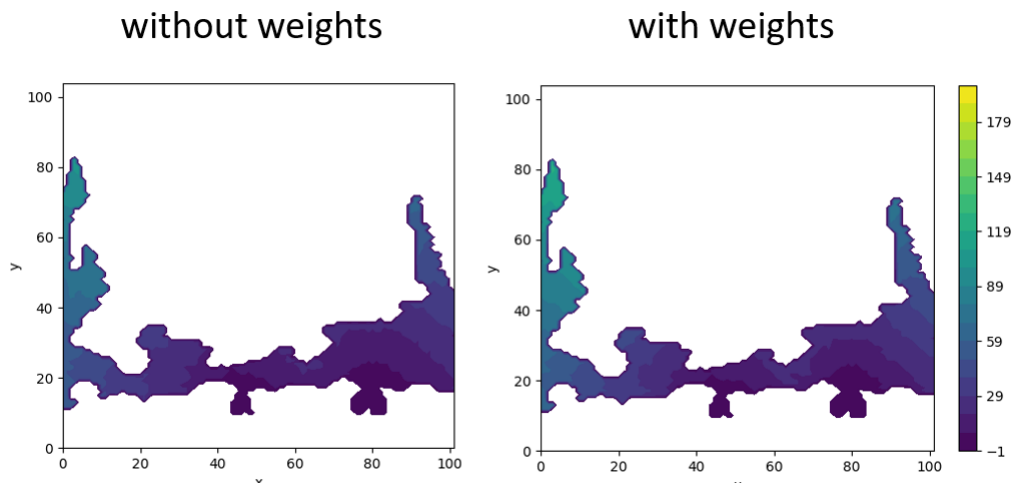

Figure 38: Contour plot with RF equal to 0.25 of long slice

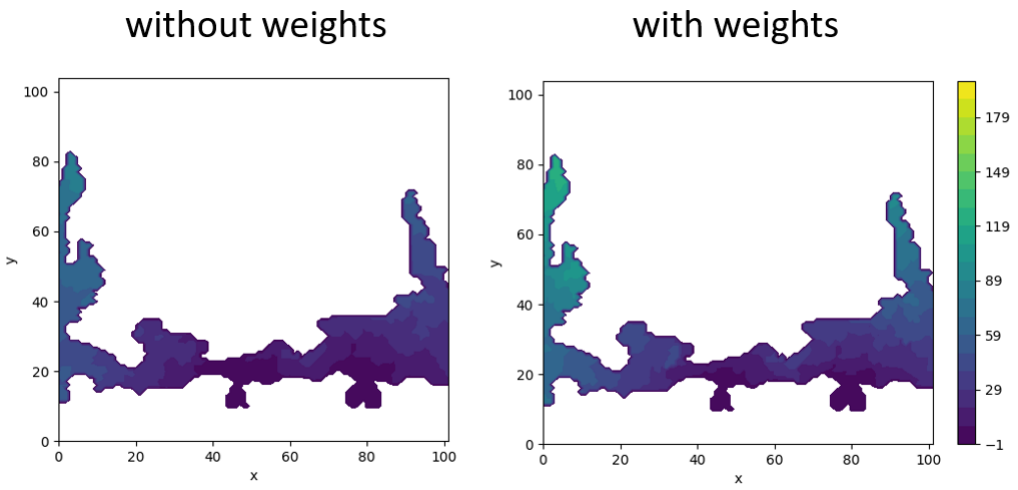

Figure 39: Contour plot with RF equal to 0.5 of long slice

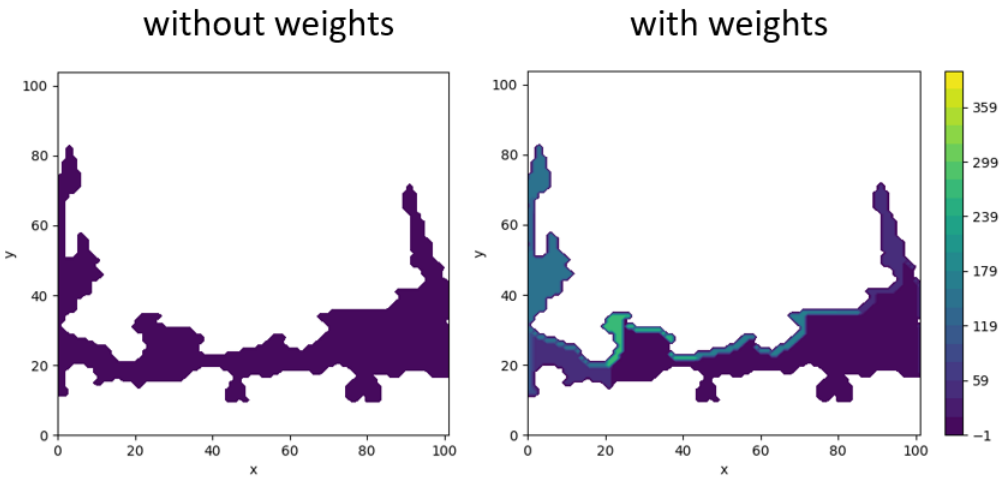

Figure 40: Contour plot with RF equal to 1 of long slice

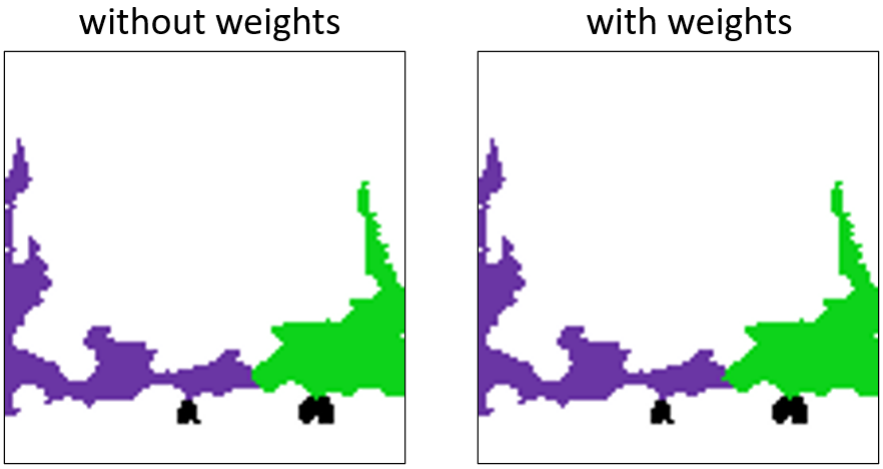

Figure 41: Voronoi diagram with RF equal to 0.25 of long slice

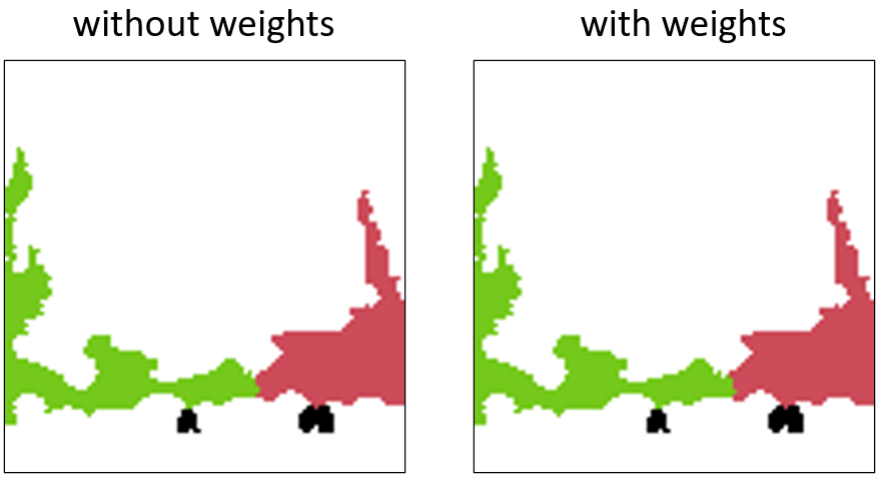

Figure 42: Voronoi diagram with RF equal to 0.5 of long slice

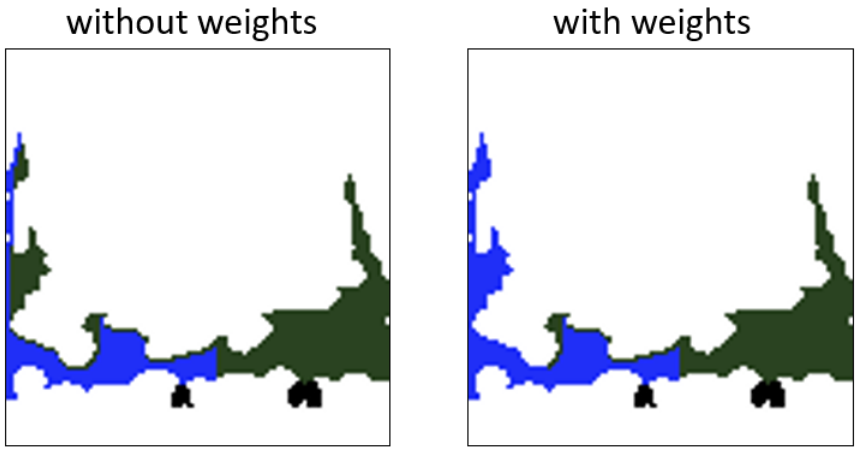

Figure 43: Voronoi diagram with RF equal to 1 of long slice

4 Conclusions

In this work two algorithms to compute the DT of a gmap have been designed and implemented. Moreover, such algorithms have been used to compute the diffusion distance and Voronoi diagrams to study the gas exchange process that occurs in plant leaves.

First, the problem of computing the DT of a gmap has been exposed, analyzing the theoretical background and the current state of the art. In fact, we analyzed the definition of gmap and its operations. Then we defined and described the DT operator, analyzed how it can be generalized to work with graphs and gmaps. Finally, the gas exchange process and the Watergate project have been described.

Subsequently, the two solutions implemented to compute the DT have been described. The first one is an algorithm that can be used to compute the DT of unweighted gmap, and it's an adaptation of the BFS algorithm for graphs. Then we defined the second algorithm, used for weighted gmaps, that is based on Dijkstra's algorithm. Both algorithms have been naively defined and then generalized to work with an arbitrary unit to propagate distance (darts, or an arbitrary i-cell) and to manage obstacles. Then the two algorithms have been used in a pipeline to compute the diffusion distance and Voronoi diagrams. The pipeline has 6 steps. First the size of the image can be reduced, then connected components have to be labeled and a gmap has to be built from the image. The gmap can be optionally reduced. At this point the DT is obtained, and the diffusion distance value can be computed. Finally, Voronoi diagrams can be generated.

Experiments have been made to analyze the trade-off between accuracy and speed in computing the diffusion distance reducing the gmap varying the reduction factor. Results show that up to a value of 0.5 for the reduction factor, acceptable estimates of the diffusion distance can be obtained, while if the RF is equal to 1 the speed increases considerably, but the error is too high. We tested both weighted and unweighted algorithms, observing that neither works better in every situation. By using weights the value of the distance transform is overestimated, while by not using them the value is underestimated.

5 Future work

Algorithms parallelization It could be possible to parallelize the algorithms to improve performance on parallel architectures. In particular, we have analyzed methods to parallelize the algorithm to compute the DT of unweighted gmaps. For example the very efficient **Jump Flooding Algorithm (JFA)** [12], that is defined for images, exploits the GPU and has a constant time complexity in the number of seeds and a logarithmic time complexity in the length of the side of the image. Unfortunately the JFA doesn't work in the presence of obstacles, so we cannot use this approach to compute the DT, but maybe it could be possible to derive a similar algorithm with the same time complexity. Instead, **parallel breadth-first search algorithms** defined for graphs like the ones presented in [13] and [14] could be adapted to work also for gmaps to compute the DT.

Efficient gmap implementation As has been written in Section 1.1 the gmap implementation has been improved to especially reduce memory consumption. For very demanding applications it can be considered to further improve the implementation. One way to do that is to re-implement the entire data structure or at least its core part **using a more efficient programming language**, like C or C++. In this way not only the memory consumption can be reduced, but also the time performance can be greatly improved. As suggested by Prof. Jiří Hladůvka of PRIP Lab a way to do that could be to define the interface in python and then write C functions for the core parts, that can be called using **Python bindings**[8].

Improve segmentation To better study the gas exchange process, it could be useful to improve the segmentation of the images before computing the DT. In fact, as can be observed in Fig. 23 cells in the raw image are distinct while in the segmented one, cells, in black, form unique huge regions. We want to get an image with **separated cells**, to be able to compute the distance from stomata to all of them. At present, **the contiguous regions act as obstacles** and prevent the distance from propagating throughout the leaf.

[8] *Ctypes — a foreign function library for python*, `https://docs.python.org/3/library/ctypes.html`, Accessed: 03-02-2022.

Propagation speed **Bottlenecks and the size of stomata** can be used to retrieve information about the propagation speed, that is another important parameter analyzed when studying the gas exchange process. The DT operator can be used also in this case, to obtain for example infomation about size and geometry of the bottlenecks.

Acknowledgments

This work would not have been possible without the support of many people.

Many thanks to my supervisors Professor Walter Kropatsch and Professor Jiří Hladůvka, for their patience, guidance and support. I have benefited greatly from their wealth of knowledge. I am indebted to all members of PRIP, especially to Majid Banaeyan who provided invaluable feedback even though he was not involved in the project. Thanks to the University of Salerno for awarding me a scholarship, providing me with the financial means to complete this project. Thanks to Technische Universität Wien who welcomed me as a student during my time in Erasmus.

Thank you to Luca Boccia and Carmine Napolano, who have accompanied me in this experience. I couldn't have wished for better companions.

Thank you to my colleagues and friends, especially to Davide Cafaro, Luca Cuomo and Emanuele D'Arminio with whom I spent my entire university period.

Most importantly, I am grateful for my family's unconditional, unequivocal, and loving support.

References

[2] P. L. Guillame Damiand, *Combinatorial maps: Efficient Data Structures for Computer Graphics and Image Processing*. CRC Press, 2014.

[3] P. Lienhardt, "Topological models for boundary representation: A comparison with n-dimensional generalized maps," *Computer-Aided Design*, vol. 23, no. 1, pp. 59–82, 1991, ISSN: 0010-4485. DOI: https://doi.org/10.1016/0010-4485(91)90082-8. [Online]. Available: %5Curl%7Bhttps://www.sciencedirect.com/science/article/pii/0010448591900828%7D.

[4] ——, "N-dimensional generalized combinatorial maps and cellular quasi-manifolds," *International Journal of Computational Geometry and Applications*, vol. 4, no. 3, pp. 275–324, 1994, ISSN: 0218-1959. [Online]. Available: %5Curl%7Bhttps://www.worldscientific.com/doi/abs/10.1142/S0218195994000173%7D.

[6] W. G. Kropatsch, "Distance transforms on maps," Jul. 2021.

[10] R. Lotufo, A. Falcão, and F. Zampirolli, "Fast euclidean distance transform using a graph-search algorithm.," Jan. 2000, pp. 269–275. DOI: 10.1109/SIBGRA.2000.883922.

[12] G. Rong and T.-S. Tan, "Jump flooding in gpu with applications to voronoi diagram and distance transform," in *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, ser. I3D '06, Redwood City, California: Association for Computing Machinery, 2006, pp. 109–116, ISBN: 159593295X. DOI: 10.1145/1111411.1111431. [Online]. Available: %5Curl%7Bhttps://doi.org/10.1145/1111411.1111431%7D.

[13] R. Berrendorf and M. Makulla, "Level-synchronous parallel breadth-first search algorithms for multicore and multiprocessor systems," Venice, Italy: Proc. Sixth Intl. Conference on Future Computational Technologies and Applications (FUTURE COMPUTING 2014), 2014, pp. 26–31.

[14] C. E. Rodrigues Alves, E. Cáceres, F. Dehne, and S. Song, "A parallel wavefront algorithm for efficient biological sequence comparison," May 2003, pp. 249–258, ISBN: 978-3-540-40161-2. DOI: 10.1007/3-540-44843-8_27.

List of Figures

List of Tables