University of Groningen

A Graph Formalism for Time and Memory Efficient Morphological Attribute-Space Connected Filters

Babai, Mohammed; Chowdhury, Ananda S.; Wilkinson, Michael H. F.

Published in:
Mathematical Morphology and Its Applications to Signal and Image Processing

DOI:
[10.1007/978-3-030-20867-7_22](https://doi.org/10.1007/978-3-030-20867-7_22)

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2019

[Link to publication in University of Groningen/UMCG research database](#)

Citation for published version (APA):

Babai, M., Chowdhury, A. S., & Wilkinson, M. H. F. (2019). A Graph Formalism for Time and Memory Efficient Morphological Attribute-Space Connected Filters. In B. Burget, A. Kleefeld, B. Naegel, N. Passat, & B. Perret (Eds.), *Mathematical Morphology and Its Applications to Signal and Image Processing* (pp. 281-294). (Lecture Notes in Computer Science; Vol. 11564), (Image Processing, Computer Vision, Pattern Recognition, and Graphics; Vol. 11564). Springer International Publishing. https://doi.org/10.1007/978-3-030-20867-7_22

Copyright

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

Take-down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

A Graph Formalism for Time and Memory Efficient Morphological Attribute-Space Connected Filters

Mohammad Babai¹(✉), Ananda S. Chowdhury², and Michael H. F. Wilkinson¹

¹ University of Groningen, Groningen, The Netherlands

{m.babai,m.h.f.wilkinson}@rug.nl

² Jadavpur University, Kolkata, India

as.chowdhury@jadavpuruniversity.in

Abstract. Attribute-space connectivity has been put forward as a means of improving image segmentation in the case of overlapping structures. Its main drawback is the huge memory load incurred by mapping a N -dimensional image to an $(N + \dim(A))$ -dimensional volume, with $\dim(A)$ the dimensionality of the attribute vectors used. In this theoretical paper we introduce a more space and time efficient scheme, by representing attribute spaces for analysis of binary images as a graph rather than a volume. Introducing a graph formalism for attribute-space connectivity opens up the possibility of using attribute-space connectivity on 3D volumes or using more than one attribute dimension, without incurring huge memory costs. Furthermore, the graph formalism does not require quantization of the attribute values, as is the case when representing attribute spaces in terms of $(N + \dim(A))$ -dimensional discrete volumes. Efficient processing of high dimensional data produced by multi-sensor detection systems is another advantage of application of our formalism.

Keywords: Image segmentation · Connectivity · Attribute-spaces · Graph representations · Graph morphology · Morphological filtering

1 Introduction

In many image-processing applications, one needs to segment the image into disjoint connected components. This task may fail when the structures overlap, because segmentation aims to provide a partition rather than a cover of the image domain. A solution to this problem was proposed in [24], by introducing the concept of connectivity in higher dimensional spaces, known as *attribute spaces*. In this scheme, which was only defined for binary images, each foreground pixel (x, y) in the original image is mapped to a non-empty set $\{(x, y, a_i)\}$, with a_i indicating the i^{th} attribute value assigned to the pixel, with i from some index set I . Transforming the image into a higher dimensional space could theoretically be a solution to segment overlapping structures into disjoint segments, as shown

Fig. 1. An example of attribute-space connectivity: (Left) Original image containing a single connected component; (right) mapping to a 3D orientation space separates the horizontal from the vertical bar.

in Fig. 1, where all foreground pixels in a 2D binary image are mapped to a 3D orientation space, by assigning one or more orientations to each pixel, e.g. using a scheme similar to that in [8].

The theory put forward in [24] can readily be extended to deal with 3D volumes as input, but expansion to 4D would be very costly in terms of memory use. Likewise, the theory can also be extended to more than one attribute dimension, e.g. orientation and curvature, which again would lead to 4D data hypercubes, even if we restrict ourselves to 2D inputs.

There is a further drawback to representation of the images or volumes transformed into the attribute space. List of attribute values with a pre-defined length for all pixels of an image (multi-dimensional arrays) requires prior discretization of the attribute values. On binary images, many attributes, such as width might be by nature discrete, so do not pose a problem. However, other attributes, such as orientation and curvature are not necessarily discrete, which means we need to quantize the data for representation in an N -dimensional array.

We therefore turn to mathematical morphology on graphs [23], which has recently come under renewed attention [5, 6, 13]. An image can be interpreted as a graph where vertices are represented by the pixels and the edges are defined by the adjacency relations on those pixels. In a recently submitted paper [1] we made a first attempt at using the theory of attribute-space connectivity in the domain of graphs. In the current work, we improve upon the previous attempt, by assigning attribute values not just to vertices, but also to the edges themselves, which allows a far simpler formalism, and obviates the need of using tunable parameters such as thresholds, which was required in the previous method.

The rest of the paper is organised as follows: The basics of connectivity and filtering are briefly described in Sect. 2. In Sect. 3 we explain in short the basics of attribute-spaces on graphs, translating original definition of attribute-space transform pairs to the graph domain. We describe our method to compute connected components in Sect. 4 and finally, in Sect. 5 we discuss our method and future work.

2 Basic Concepts of Connectivity and Filtering

In mathematical morphology images are usually represented as subsets X of some image domain M [19]. Members of X are the foreground pixels, and connectivity between pixels is defined by 4 or 8 neighbourhood relationships. Connected components are usually defined using path connectivity, but a more general framework is that of *connectivity classes* or *connections*. A connectivity class $\mathcal{C}$ is the family of all connected subsets of M , with the following properties:

- $\emptyset \in \mathcal{C}$.
- Singletons $\{x\} \in \mathcal{C}$ for all $x \in M$.
- For any family $\{X_i\} \subseteq \mathcal{C}$, $\bigcap_i X_i \neq \emptyset$ implies $\bigcup_i X_i \in \mathcal{C}$.

as defined in [20]. Each connectivity class is associated with a family of connectivity opening Γ_x , with one associated with each $x \in M$.

$$\Gamma_x(X) = \bigcup \{C \in \mathcal{C} \mid x \in C \wedge C \subseteq X\}. \quad (1)$$

The function $\Gamma_x(X)$ returns the connected component of X to which x belongs, if $x \in X$ and the empty set otherwise.

These notions extend readily to complete lattices [20], and therefore also to a graph formalism, in which a binary image X is represented as a subgraph $G = (V, E)$ of some maximal graph $G_M = (V_M, E_M)$ with $V \subseteq V_M$, $E \subseteq E_M$ and the set of vertices $V_M \subseteq M$ and a set of edges $E_M \subseteq (M \times M)$. The lattice $\mathcal{L}_G$ associated with G_M in this way is simply the set of all subgraphs of G_M , with the empty graph $\mathbf{0}$ as global infimum and G_M as supremum, and the partial order relation $a \preceq b$ mean a is a subgraph of b . All that has really changed is making the neighbourhood relationship explicit in the form of the set of edges E . The vertices of G represent the pixels and the edges are determined by the neighbourhood relations. Each vertex $v \in V$ has a maximum of 8 edges, in a 2D image with 8-connectivity scheme. The spatial coordinates, grey or colour value or any other local pixel attributes are stored in the nodes, depending on the graph embedding or the properties of the input image; an example method to generate a graph from an image is illustrated in Algorithm 2.1. The graph G can be constructed in $O(N)$ time, where N is the total number of pixels in the image. Note that in general graph nodes could be of an arbitrary dimension and can have arbitrary and non-uniform neighbouring relations.

Algorithm 2.1 Create a graph G from an image.

Input: (Image img)

Output: (Graph G)

- 1: **for all** p (pixels) $\in \text{img}$ **do**
 - 2: Create node g and store all attributes of p in g .
 - 3: Add indices of all neighbours of p to neighbour-list of g .
 - 4: Insert g into G .
 - 5: **end for**
-

The connectivity classes described in [17, 20, 24] can be used to define connectivity. Every connectivity class $\mathcal{C} \subseteq \mathcal{L}_G$ can be considered as the family of all connected elements of a lattice; in this case the family of all connected subgraphs of some maximal graph $G_M = (V_M, E_M)$. Usually a graph $G = (V, E) \preceq G_M$ is considered connected if $\forall a, b \in V$, there is a sequence $(x_1, \dots, x_n) \subseteq V$ with $x_1 = a$ and $x_n = b$ and all edges $e_i = (x_i, x_{i+1}) \in E$. We can define connected components $\{C_i\}$ of any graph $G = (V, E)$ in the usual way, as connected subgraphs of G of maximal extent, i.e. $\{ \forall C_i, \nexists (C \succ C_i) : C \preceq G \wedge C \in \mathcal{C} \}$. Connectivity openings can be defined in a similar way, by allowing connectivity openings marked more generally by any member of the connectivity class $\mathcal{C}_G$, not just the family of singleton graphs $\mathcal{S}_G$ which contains all graphs $g \in \mathcal{L}_G$ of the form $(\{v\}, \emptyset)$ or $(\emptyset, \{e\})$, i.e. both singleton vertices and singleton edges. We consider all elements of $\mathcal{S}_G$ connected. A binary connected opening Γ_x on $G \in \mathcal{L}(G)$ for any connected subgraph x of G_M can be defined as:

$$\Gamma_x(G) = \bigvee \{C \in \mathcal{C} \mid x \preceq C \wedge C \preceq G\}. \quad (2)$$

$\Gamma_x(G)$ is the connected component in which $x \preceq G$ is contained and $\mathbf{0}$ otherwise. Generalisations such as second-generation connectivities as developed on images [3, 15, 20] should also extend quite naturally to the graph framework.

Attribute filters form a class of the connected filters. Analogous to binary images, we can define connected filters [18], and in particular attribute filters [4] which remove or select connected components that fulfil a certain criterion T as:

$$\Gamma^T(G) = \bigvee_{x \in \mathcal{S}_G} \Gamma_T(\Gamma_x(G)). \quad (3)$$

with Γ_T the *trivial filter*, which when applied to some graph C , returns C if it meets some criterion T , and $\mathbf{0}$ otherwise. Usually, T is of the form $T(C) = \text{Attr}(C) \geq \tau$, with the property of being increasing; τ is a threshold value [4] and $\text{Attr}(C)$ is an attribute of C .

As can be seen, by casting graph-based connected filtering into a lattice framework, connectivity and connected filtering readily extends to the graph case. Extensions such as hyperconnections [16, 27], which have also been cast into lattice formalism should also be straightforward. The same does not hold true for attribute-space connections, for which only a set-based formalism exists. In the following section we will extend the formalism to the lattice of graphs $\mathcal{L}_G$.

3 Attribute-Space Connectivity

In [24] attribute-space connectivity and filters were introduced in the case of binary images. The core idea is to transform the image into a higher-dimensional domain in which overlapping structures are readily separated. An image X is transformed into a higher dimensional ‘‘attribute space’’ $(M \times A)$, with A the attribute dimension. The latter represents some local measure of the image, such as width or orientation. This is done by mapping every pixel $x \in X$ to one or more

tuples $(x, a_i) \in (X \times A)$ using some operator $\Omega : \mathcal{P}(M) \rightarrow \mathcal{P}(M \times A)$; $\mathcal{P}$ denotes the power set and i is from an index set I . Note that $\Omega(X)$ yields a binary image in $(M \times A)$ and A is typically a subset of $\mathbb{Z}$ or $\mathbb{R}$. These separated structures can then be filtered in the $(M \times A)$ space using an appropriate attribute filter on the connected components. After application of the selected filters the results could be projected back onto M by the inverse operator $\Omega^{-1} : \mathcal{P}(M \times A) \rightarrow \mathcal{P}(M)$. In summary, the required properties [25] of the pair (Ω, Ω^{-1}) are:

1. $\Omega(\emptyset) = \emptyset$,
2. $\forall x \in M : \Omega(\{x\}) \in \mathcal{C}_{M \times A}$,
3. The mapping $\Omega : \mathcal{P}(M) \rightarrow \mathcal{P}(M \times A)$ satisfies the condition that for all $X \in \mathcal{P}(M)$, each $x \in X$ is mapped to at least one point $(x, a) \in \Omega(X)$, with $a \in A$,
4. The inverse mapping $\Omega^{-1} : \mathcal{P}(M \times A) \rightarrow \mathcal{P}(M)$ satisfies the condition that for all $Y \in \mathcal{P}(M \times A)$, each $(x, a) \in Y$ is projected to $x \in \Omega^{-1}(Y)$. In other words, every tuple is mapped to the actual pixel in the original image domain (M) ,
5. $\forall X \in \mathcal{P}(M) : \Omega^{-1}(\Omega(X)) = X$,
6. Ω^{-1} is increasing; $\{\forall X, Y \in \mathcal{P}(M \times A) : X \leq Y \implies \Omega^{-1}(X) \leq \Omega^{-1}(Y)\}$.

Since axiom 4 defines Ω^{-1} to be a projection and thus being increasing, the axiom 6 is redundant [25], and we will therefore leave it out in the following discussion. As we have discussed earlier, a naive application of this idea incurs a huge memory overhead, which in turn increases computational cost.

Alternatively, we simply only duplicate the required vertices and edges of G , to form a new graph $G^A = (V^A, E^A)$ with $V^A = (V \times A)$, and $E^A \subseteq (V^A \times V^A)$. As our recently submitted paper [1], we denote each element of V^A as $v_{ij}^A = (v_i, a_j)$, with i and j from index sets over V and A respectively. For every v_{ij}^A , a_j is referred to as the node's *attribute value*. In our previous work edges in G^A are defined as

$$E^A = \{((v_{ij}^A, v_{kl}^A) \in V^A \times V^A \mid (v_i, v_k) \in E) \wedge (||a_j - a_l|| < D)\}, \quad (4)$$

with D some threshold. Thus nodes in G^A are connected by an edge, if their counterparts in graph G are connected by an edge and their difference in attribute value is small enough. This has a slight problem in that some edges might be removed entirely by Ω , and unless certain slightly artificial measure were taken, back-projection by Ω^{-1} would not restore them. This would imply that $\Omega^{-1}(\Omega(G)) \neq G$ for some graphs G , which is clearly undesirable.

Here, we choose a more general approach, in that transformation operator Ω assigns not only one or more attributes to the vertices, but also one or more *pairs* of attributes to each edge $e \in E$. Thus, we can write

$$\Omega(G) = (\Omega_V(G), \Omega_E(G)) \quad (5)$$

Algorithm 3.1 Compute Attribute Space for a graph G .

Input:(Graph G)

Output:(Graph G^A)

```

1: for all  $x \in G$  do
2:   if ( $x$  is foreground) then
3:     Compute  $(x, a_i)$  and  $(e, a_{1j}, a_{2j})$ .
4:     Store  $(a_i)$  in  $x$  and  $(a_{1j})$  and  $(a_{2j})$  in  $e$ .
5:   end if
6: end for

```

with $\Omega_V : \mathcal{L}_G \rightarrow \mathcal{P}(M \times A)$ essentially the same operator on vertices as in [1], which yields V^A , and $\Omega_E : \mathcal{L}_G \rightarrow \mathcal{P}((M \times A) \times (M \times A))$ a mapping from a single edge in the original graph to one or more edges in $G^A = \Omega(G)$. As before, $\mathcal{P}$ denotes the power set. We assume the result G^A is subgraph of some maximal graph G_M^A embedded in the attribute space, with corresponding lattice of subgraphs $\mathcal{L}_M^A$.

Edges e^A in these latter graphs can be denoted either as pairs of vertices in the attribute domain (v_1^A, v_2^A) or equivalently, as triples (e, a_1, a_2) , with $e \in M \times M$, and $a_1, a_2 \in A$. Therefore Ω_E may assign different attribute values to each end-point of the edges in G^A ; the determined value depends on the local properties of the attached structures. This method is illustrated in Algorithm 3.1; note that adding an attribute a_i internally to a nodes' attributes list is equivalent to splitting the node. One can construct G^A from G in $O(|V|)$ time.

We can readily incorporate the mapping of (4) into this mechanism, but at the same time allow disconnected edges to appear in the transformed graph G^A . Attribute values are assigned to each end-point of every edge connecting active regions of the graph G ; these values may differ per end-point of the edges. This is equivalent to creating a copy of each edge for each attribute value. This operation allows for an edge to be active in one attribute-space while not connected

Fig. 2. Loose edges in attribute space: (left) graph G showing two diamond-shaped structures linked in M by the red edge; (right) width-space projection G^W in which all edges and vertices are assigned the value of the diameter of the largest inscribed city-block disk in which they fit. The red edge does not fit into any disk, and is assigned a zero value, disconnecting the structures in width space. Without filtering, back projection of G^W onto the original space yields the original graph. Note that we use L_1 norm to determine the width-based attribute for each node. (Color figure online)

(active) in another attribute-space as a consequence of not fulfilling a certain criterion; this simplifies the inverse operator Ω^{-1} . Ω^{-1} converts G^A to G by merely projecting the attribute dimensions for all vertices and edges in G^A to the original domain of G . This effect is illustrated in Fig. 2, which uses a width space transform Ω_W (see Sect. 3.1). The centre edge linking the two diamond shapes in the original graph G is assigned a low width at either side, resulting in it being disconnected in $G^W = \Omega_W(G)$. Without filtering G^W , the back projection operator Ω_W^{-1} will simply result in the original graph G , as desired.

The operator $\Omega(G) = (\Omega_V(G), \Omega_E(G))$, in Eq. 5, is basically the same as $\Omega(X)$ in the case of binary images. It is merely split in two operators, one to determine the mapping for the vertices and one for the mapping of the edges, respectively. Thus, the previously discussed set of required properties apply to $\Omega(G)$ as well. In summary $\Omega(G)$ has the following properties:

1. $\Omega(\mathbf{0}) = \mathbf{0}$,
2. $\forall G \in \mathcal{S}_G : \Omega(G) \in \mathcal{C}_{M \times A}$,
3. $\Omega : \mathcal{L}_G \rightarrow \mathcal{L}_G^A$ is a mapping such that for any $G = (V, E) \in \mathcal{L}_G$ each $v \in V$ is mapped to at least one corresponding element $(v, a) \in \Omega_V(G)$, and each $e \in E$ is mapped to at least one corresponding element $(e, a_1, a_2) \in \Omega_E(G)$,
4. $\Omega^{-1} : \mathcal{L}_G^A \rightarrow \mathcal{L}_G$ is a mapping such that for every $G^A = (V^A, E^A) \in \mathcal{L}_G^A$, each $(v, a) \in V^A$ is projected onto $v \in \Omega_V^{-1}(G^A)$, and each $(e, a_1, a_2) \in E^A$ is projected onto $v \in \Omega_E^{-1}(G^A)$,
5. $\Omega^{-1}(\Omega(G)) = G, \forall G \in \mathcal{L}_G$

Using these properties, one can define an attribute-space connection $\mathcal{A} \subseteq \mathcal{L}_G$ generated by the transformation pair (Ω, Ω^{-1}) and connection $\mathcal{C}_A$ on $\mathcal{L}_G^A$ as:

$$\mathcal{A}_\Omega = \{C \in \mathcal{L}_G \mid \Omega(C) \in \mathcal{C}_A\}. \quad (6)$$

Using the definitions above one can define attribute-space connected filters as:

$$\Phi^A = \Omega_A^{-1}(\Phi(\Omega_A(G))). \quad (7)$$

With $G \in \mathcal{L}_G$ and $\Phi : \mathcal{L}_G^A \rightarrow \mathcal{L}_G^A$ a connected filter and $(\Omega_A, \Omega_A^{-1})$ a transformation pair for the attribute A .

3.1 Width-Based Attributes

Local width is an attribute that could be used to transform a binary image $X \in \mathcal{P}(M)$ to $Y \in \mathcal{P}(M \times A)$, with $A \in \mathbb{Z}^+$. The corresponding mapping Ω_W could be implemented using an opening transform defined by granulometry $\{\theta_r\}$ [11]. Here, each operator $\theta_r : E \rightarrow E$ is an opening with the structuring element Θ_r . The opening transform Ω_W on the binary image X could be defined as follows:

$$\Omega_W(X) = \max\{r \in A \mid x \in \theta_r(X)\}. \quad (8)$$

Fig. 3. An example of assigned width-based attributes to the pixels of a binary image. (left) Original image. (middle) transformed image to the width-based attribute space. Each pixel is assigned the radius of the largest ball. Darker colour indicates smaller radius. (right) partitioning of X using a threshold τ . Because of abrupt changes in r , X is split in two components.

If $\theta_r(X) = X \circ \Theta_r$ with $\circ$ denoting a structural opening with a ball-shaped structuring element Θ_r of radius r , an opening transform assigns the radius of the largest circle to each $x \in X$ such that $x \in X \circ \Theta_r$. We can now write the mapping $\Omega_W : \mathcal{P}(E) \rightarrow \mathcal{P}(E \times \mathbb{Z}^+)$ as:

$$\Omega_W(X) = \{(x, Y) | x \in X\} \text{ with } Y = \Omega_W(x) \quad (9)$$

and Ω_W^{-1} as:

$$\Omega_W^{-1}(Y) = \{(x \in E) | (x, y) \in Y\}. \quad (10)$$

An example of a width space transformation is shown in Fig. 3. In the case of graph representation of an image, the original operator Ω_W consists of a pair $(\Omega_{W,V}, \Omega_{W,E})$. $(\Omega_{W,V})$ assigns width values to each node and $(\Omega_{W,E})$ determines a width-based attribute value for each end-point of each edge. In the example shown in Fig. 3 we used L_∞ to determine the value of r for each pixel, whereas in Fig. 2 the city-block distance (L_1) is used to compute the radius of the largest disk; note that the edges of the left diamond receive a value of $r = 3$ and the right diamond $r = 4$. The red edge receives a smaller attribute value at both ends; thus it is not connected in some width-based spaces.

3.2 Orientation Attributes

Using the binary-image based attribute-space representation, an operator $\Omega_\beta : \mathcal{P}(M) \rightarrow \mathcal{P}(M \times A)$, with $A = \{\beta_1, \beta_2, \dots, \beta_n\}$, representing N different orientations, can be defined which assigns one or more orientation-attributes β_i to each active node (pixel) $x \in X$ [24]. The result is a binary image $f : (M \times A) \rightarrow \{0, 1\}$, which would be the indicator function of the set $\Omega_\beta(X) \subseteq M \times A$.

Fig. 4. An example of assigned orientation attribute to pixels of an image. Each attribute-space corresponds to a attribute-value and contains the entire image including the background pixels. (left) original image; (middle) pixels with $\beta = 90^\circ$; (right) pixels with $\beta = 0^\circ$. The central pixel is active in all orientations while the other pixels have only one orientation-attribute.

A naive implementation of this idea would simply create n copies of the original image X (a 2D array storing the pixels), and set the pixels $x \in \Omega_\beta(X)$ to active. Figure 4 illustrates this idea by showing a few slices of such a volume. This straightforward approach has a significant impact on the required memory size and computational intensity of the algorithm. Alternatively we represent the binary image as a subgraph G of G_M , in which we only include the active nodes and relevant edges, and omit all others. As discussed before, the original operator Ω_β now consists of a pair $(\Omega_{\beta,V}, \Omega_{\beta,E})$, where $\Omega_{\beta,V}$ creates one or more copies of all the vertices of G to the new graph G^β . However, rather than assigning discrete values $\beta_1, \beta_2, \dots, \beta_n$, we can now assign any value in the domain $[0, 180)$. A similar mapping is applied to the edges by $\Omega_{\beta,E}$. In the example from Fig. 1, this means only the centre pixel of the cross needs to be copied more than once, so only a tiny memory penalty is incurred. A similar scheme is applied to the edges connecting the pixels; each edge receives an orientation value for each end-point; one could use Algorithm 3.1 to compute $G^\beta = (V^\beta, E^\beta)$. By way of comparison, the original approach would require a memory cost of $O(|G_M|n)$, with n the number of values β might take, and $|G_M|$ the storage size of G_M , whereas the new approach requires $O(|G|n_i)$, with n_i the number of intersections of components. This also means that all subsequent processing is on a far smaller graph, which reduces computational cost.

4 Connected Components Analysis, Higher-Dimensional Attributes and Filtering

Application of the discussed operator Ω^A on G constructs G^A ; one can then apply connected component analysis and attribute filtering on G^A and mapping the results to the original domain using Ω^{-1} . Connected component analysis on graphs can be achieved efficiently, for example, by applying Tarjan’s Union-Find algorithm [21, 22], in pseudo-linear time complexity. This represents disjoint sets as a forest of spanning trees over G^A . Initially, all vertices in G^A are considered singleton trees. The algorithm then processes all edges in G^A . The algorithm

merges the trees to which the vertices are linked by each edge inspected. This is done by letting the root pointer of one tree point to the root node of the other. Because not all edges link two vertices in our case, we only perform this merge if the edge actually links to existing vertices of G^A . During merger, path compression and union-by-rank are used to keep root paths as short as possible. For details see [21, 22], and for more recent papers on its application to connected filtering check [9, 10, 12].

In the preceding discussion we have focused on regular 4- or 8-connectivity. However, using graph representation allows for processing patterns with non-uniform neighbouring definitions in high dimensional data. For example consider the observable patterns in point clouds produced by multi-sensor detector networks. In those structures, different nodes have a varying number of neighbours and connectivity structures based on the sensor densities in different locations. Furthermore, the notion of connectivity has been generalised [3, 15, 17, 20], allowing objects separated by small gaps (caused e.g. by noise) to be considered connected. It has been shown that these forms of connectivity can also be implemented in a graph formalism [14, 26]. The memory and compute loads do tend to increase, but only by a constant factor.

During the union-find process, we can compute various higher-order derived attributes as the components grow in size at each merge. One of the easiest attribute is the area (or volume in 3D), which simply involves counting the number of vertices involved [10]; at each node $g \in G^A$, we store a record to hold the value of the largest area to which g belongs. When handling thin and elongated structures, one of the obvious attributes to compute is the curvature of the processed patterns. One can determine the curvature for a set of at least three pixels. These attributes are updated while the connected components grow. The higher order attributes could serve not only for filtering or segmentation but also as criterion for merging connected components, specially when small gaps are allowed between the components. Because we only copy nodes and edges to an attribute-space if they are active in that particular space, the size of the required memory remains smaller than number of nodes and edges times number of attributes. Using this scheme, some nodes may contain a vector of attributes and the others may have only a few. For example, the central node in Fig. 1 left panel may contain width and multiple orientation attributes while all other pixels contains at most two attribute values namely the width and a single value for the local orientation. Once we have transformed G to G^A by determining all attributes $a \in A$ for vertices and edges of G , connected filters could be defined to select or discard connected components based on the attribute values. A possible method for connected component analysis is shown in Algorithm 4.1.

We have applied this formalism to process the data produced by the Straw Tube Tracker (STT) of the PANDA experiment [7], to reconstruct sub-atomic particle tracks [1]. This detector (STT) is an irregular-graph containing more than 4000 tubes with a varying length between 150 cm down to a few centimetres. The tubes are arranged in layers around the beam pipe (z -axis). A number of these layers are tilted with $+3$ or -3° with respect to the z -axis. Because of

Algorithm 4.1 Connected component analysis.

Input: (Graph G^A , a set of criteria T .)

Output: (Graph $G' \in G$. The output is the maximum sub-graph G' of G fulfilling the criterion T .)

```

1: for all ( $v' \in V^A$ ) and ( $e' \in E^A$ ) do
2:   if  $v'$  and  $e'$  fulfil  $t_i$  then
3:     Add  $v'$  and  $e'$  to  $G'$ 
4:   end if
5: end for
6: for all ( $(v', e') \in G'$ ) do
7:   Compute  $\Omega^{-1}(v', e')$ 
8: end for

```

this tilt structure, the number of neighbours of the tubes varies between 2 and 22 tubes. When a charged particle passes through the STT, it turns a subset of the tubes on and produces a cloud of points containing a binary image in 3D. Reconstructing paths using these points needs to be performed at run-time (at 20 MHz). Figure 5 shows an example of a pattern produced with such a multi-sensor detection system. We have computed the orientation attribute for all active tubes in the graph. One can observe that each node has a few local orientation attributes and the value of these attributes vary strongly between different nodes. Our method limits the number of copies of each node to the number of actual computed attributes for it. During the later stages the constructed connected components were merged on the basis of their distances where the length of the connected components attached to each edge was used to weight the preferred path growth-directions. Using the curvature of the line connecting the points in the space as an attribute for merging the larger components, is an obvious choice; but this requires fitting a curve through the points which is a computational intensive operation. One can evaluate the Mahalanobis distance of a point cloud as a measure for the strongest elongation direction. During the later merging operations, we have used Mahalanobis distance, as a measure for the orientation compatibility between different components; these higher order attributes were updated as the components grew. The results of this hierarchical merging based on heterogeneous conditions is shown in the panels in the second row of Fig. 5.

5 Discussion

We have introduced a new and generic graph-based formalism to construct memory-efficient methods for computing attribute-spaces and attribute-space connected filters for binary images. This formalism allows for processing larger images in higher dimensional attribute-spaces. We have studied the transformation pair $\Omega_A = (\Omega, \Omega^{-1})$ and their properties and relations, to transform graphs G to an attribute-space graph G^A and vice versa. By representing an image X as a graph, we store the determined attribute values computed using Ω_A for all active nodes in the vertices and assign attribute values to each end-point of the edges connecting those nodes. In this formalism, the nodes and the corresponding edges are copied only for actual computed attribute values;

Fig. 5. First row: (left): an example event shown in the xy -plane. The grey dots represent the detector tubes centres and the red dots the read-out positions of activated sensors. (right): the same pattern shown in the yz -plane. Second row: (left): reconstructed particle paths using an implementation of this formalism in the xy -plane. (right) reconstructed paths in the yz -plane. In the images in the second row, each colour and marker combination represents a different connected component (particle path). The differences between the reconstructed (second row) and the true (first row) particle paths in the yz -plane is partially because of the detector read-out imperfections. (Color figure online)

this mechanism allows for computing non-discrete attribute values. In general, the nodes representing the data points (pixels) at intersection points of structures in an image receive a larger number of attributes whereas all the other vertices receive only a few. We utilise this feature in our method which leads to a significant reduction of the size of the required memory for representing the image together with its copies in different attribute-spaces. This reduction makes it possible to handle larger graphs or to compute and store a larger set of distinct attributes which could be used to design heterogeneous conditions based attribute-filters for segmentation or connected component analysis. The graph-based interpretation allows for efficient and straightforward embedding of higher order attributes, such as width and curvature for each constructed connected component, which could be used for hierarchical filtering based on these attribute-values. For example, one can perform width-based merging followed by orientation-based merge operations. We have applied this method to the point-clouds produced by a multi-sensor detector to reconstruct subatomic particle trajectories. Because in this application each connected component is defined by the list of its member vertices, one can query the structure for selective processing of the available components and patterns.

Future work will consist of extending our formalism to handle grey-scale images using node-weighted and edge-weighted graphs. Further, we want to

explore the option of assigning a *range* of attribute-values to the edges and vertices in G^A rather than a single value or a pair of values. This step would enable us to process hyper-graphs [2] and design filtering and pattern analysis on hypergraph-based structures and networks. Applying this formalism to realistic and real-life graph-based data is another option to explore; this will also lead to quantitative results on comparative computation time and memory requirement.

Acknowledgement. We want to thank the Centre for Information Technology (CIT) of the Rijksuniversiteit Groningen for their support and facilitating this work.

References

1. Babai, M., Kalantar-Nayestanaki, N., Messchendorp, J.G., Wilkinson, M.H.F.: An efficient attribute-space connected filter on graphs to reconstruct paths in point-clouds. Pattern Recognition (submitted)
2. Bloch, I., Bretto, A.: Mathematical morphology on hypergraphs, application to similarity and positive kernel. Comput. Vis. Image Underst. **117**(4), 342–354 (2013)
3. Braga-Neto, U., Goutsias, J.: A theoretical tour of connectivity in image processing and analysis. J. Math. Imag. Vis. **19**, 5–31 (2003)
4. Breen, E.J., Jones, R.: Attribute openings, thinnings and granulometries. Comp. Vis. Image Understand. **64**(3), 377–389 (1996)
5. Cousty, J., Najman, L., Dias, F., Serra, J.: Morphological filtering on graphs. Comput. Vis. Image Underst. **117**(4), 370–385 (2013). <https://doi.org/10.1016/j.cviu.2012.08.016>. <http://www.sciencedirect.com/science/article/pii/S107731421200183X>, special issue on Discrete Geometry for Computer Imagery
6. Cousty, J., Najman, L., Serra, J.: Some morphological operators in graph spaces. In: Wilkinson, M.H.F., Roerdink, J.B.T.M. (eds.) ISMM 2009. LNCS, vol. 5720, pp. 149–160. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03613-2_14
7. Erni, P.C.W., Keshelashvili, I., Krusche, B., et al.: Technical design report for the panda straw tube tracker. Eur. Phys. J. A **49**, 25 (2013). Special Article - Tools for Experiment and Theory
8. Franken, E., Duits, R.: Crossing-preserving coherence-enhancing diffusion on invertible orientation scores. Int. J. Comput. Vis. **85**, 253–278 (2009)
9. Géraud, T.: Ruminations on Tarjan’s union-find algorithm and connected operators. In: Ronse, C., Najman, L., Decencière, E. (eds.) Mathematical Morphology: 40 Years On, vol. 30. Springer, Dordrecht (2005)
10. Meijster, A., Wilkinson, M.H.F.: A comparison of algorithms for connected set openings and closings. IEEE Trans. Pattern Anal. Mach. Intell. **24**(4), 484–494 (2002)
11. Nacken, P.F.M.: Image analysis methods based on hierarchies of graphs and multi-scale mathematical morphology. Ph.D. thesis, University of Amsterdam, Amsterdam, The Netherlands (1994)
12. Najman, L., Couprie, M.: Building the component tree in quasi-linear time. IEEE Trans. Image Proc. **15**, 3531–3539 (2006)
13. Najman, L., Cousty, J.: A graph-based mathematical morphology reader. Pattern Recogn. Lett. **47**, 3–17 (2014)

14. Oosterbroek, J., Wilkinson, M.H.F.: Mask-edge connectivity: theory, computation, and application to historical document analysis. In: *Proceedings of 21st International Conference on Pattern Recognition*, pp. 3112–3115 (2012)
15. Ouzounis, G.K., Wilkinson, M.H.F.: Mask-based second generation connectivity and attribute filters. *IEEE Trans. Pattern Anal. Mach. Intell.* **29**, 990–1004 (2007)
16. Perret, B., Lefevre, S., Collet, C., Slezak, E.: Hyperconnections and hierarchical representations for grayscale and multiband image processing. *IEEE Trans. Image Process.* (2011). <https://doi.org/10.1109/TIP.2011.2161322>
17. Ronse, C.: Set-theoretical algebraic approaches to connectivity in continuous or digital spaces. *J. Math. Imag. Vis.* **8**, 41–58 (1998)
18. Salembier, P., Serra, J.: Flat zones filtering, connected operators, and filters by reconstruction. *IEEE Trans. Image Proc.* **4**, 1153–1160 (1995)
19. Serra, J.: Mathematical morphology for Boolean lattices. In: Serra, J. (ed.) *Image Analysis and Mathematical Morphology, II: Theoretical Advances*, pp. 37–58. Academic Press, London (1988)
20. Serra, J.: Connectivity on complete lattices. *J. Math. Imag. Vis.* **9**(3), 231–251 (1998)
21. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. *J. ACM* **22**, 215–225 (1975)
22. Tarjan, R.E.: *Data Structures and Network Algorithms*. SIAM, Philadelphia (1983)
23. Vincent, L.: Graphs and mathematical morphology. *Signal Processing* **16**(4), 365–388 (1989). [https://doi.org/10.1016/0165-1684\(89\)90031-5](https://doi.org/10.1016/0165-1684(89)90031-5). <http://www.science-direct.com/science/article/pii/0165168489900315>, special Issue on Advances in Mathematical Morphology
24. Wilkinson, M.H.F.: Attribute-space connectivity and connected filters. *Image Vis. Comput.* **25**, 426–435 (2007)
25. Wilkinson, M.H.F.: Hyperconnectivity, attribute-space connectivity and path openings: theoretical relationships. In: Wilkinson, M.H.F., Roerdink, J.B.T.M. (eds.) *ISMM 2009. LNCS*, vol. 5720, pp. 47–58. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03613-2_5
26. Wilkinson, M.H.F.: A fast component-tree algorithm for high dynamic-range images and second generation connectivity. In: *2011 Proceedings of International Conference Image Processing*, pp. 1041–1044 (2011)
27. Wilkinson, M.H.F.: Hyperconnections and openings on complete lattices. In: Soille, P., Pesaresi, M., Ouzounis, G.K. (eds.) *ISMM 2011. LNCS*, vol. 6671, pp. 73–84. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21569-8_7