Assigning Footprints to Dot Sets: An Analytical Survey

Maximillian Dupenois and Antony Galton

School of Engineering, Computing and Mathematics, University of Exeter, UK

Abstract. While the generation of a shape, or *footprint*, from a set of points has been widely investigated, there has been no systematic overview of the field, with the result that there is no principled basis for comparing the methods used or selecting the best method for a particular application. In this paper we present a systematic classification of footprints, algorithms used for their generation, and the types of applications they can be used for. These classifications can be used to evaluate the suitability of different algorithms for different applications. With each algorithm is associated a vector of nine values classifying the footprints it can produce against a standard list of criteria, and a similar vector is associated with each application type to classify the footprints it requires. A discussion of, and a method for, the assessment of the suitability of an algorithm for an application is presented.

1 Introduction

While the generation of a shape, or *footprint*, from a set of points has been widely investigated, there has been no systematic overview of the field, with the result that there is no principled basis for comparing the methods used or selecting the best method for a particular application. In this paper we present a systematic classification of the footprints, the algorithms used, and the types of applications they can be used for. Our classification of footprints bears some similarity to the set of criteria proposed by Galton and Duckham [9] for evaluating the footprints produced by different algorithms. However, here we provide a more detailed analysis and propose a method for choosing an algorithm appropriate for a given context. It should be noted that this paper makes no attempt to evaluate these footprints, only to classify them. Using this classification we evaluate the suitability of algorithms for general applications, but this has no bearing on how 'good' a footprint is. For a discussion on the quality of a footprint with regard to perceived shape or any other cognitive criteria see [8]. The classifications of the footprints, algorithms and applications are all linked but for the sake of clarity are declared separately, with appropriate relations discussed later. While we only cover two-dimensional footprints, much of the analysis should carry over to the three-dimensional case; but it is likely that three-dimensional footprints have additional properties not covered by the present classification.

2 Definitions

The general problem under consideration is that of assigning a region-like entity to a collection of point-like entities in space. In this paper we call the former a 'footprint' and the latter 'dots'. Here we provide a brief justification for this choice of terminology.

- **Dots** We refer to dots rather than points because when considering the various algorithms it became apparent that, in addition to coordinates, the point-like entities may possess attributes such as shape, area, or velocity, any of which might be of relevance to an algorithm.
- **Footprint** What we are here calling footprints have variously been called outlines, shapes, hulls, and regions. We reject 'outline' and 'shape' as being too focussed on the boundary; while 'region' seems too general, with nothing to indicate any special relationship to the dots. 'Hull' has often been chosen to reflect the idea of a footprint as a generalisation of the convex hull, and indeed many of the algorithms are modified forms of convex-hull algorithms (e.g., Concave Hull [13]); however, the definition of a hull operator in computational geometry requires all the dots to lie within the hull, which must itself be connected [12], both of which conditions may be violated by footprint algorithms.[1] 'Footprint' is used in various fields to denote the impression of some entity,[2] and this connotation seems appropriate here.

3 Background

As mentioned in the introduction, much work has been done on the generation of footprints, but there has been surprisingly little by way of comparative analysis. Four types of analysis largely absent from the literature on footprints are:

- Analysis of the types of footprint.
- Analysis of the methods used in the algorithm, and how these methods limit the footprints produced.
- Analysis of the algorithm type, e.g., whether it requires some form of preprocessing on the data set.
- Analysis of the context that the algorithm was created for.

An early, and much-referenced, paper on the subject is by Edelsbrunner et al. [5], who present a method for creating footprints from a point set.[3] The method produces straight-line graphs called α-shapes, obtained from a generalisation of the convex hull. For a set S the convex hull can be considered to be the intersection of all closed half-planes that contain all the points of S. The *α-hull*

[1] A hull operator is also required to be idempotent; it is not clear what this could mean in the case of an operator which generates a region from a finite set of points.

[2] E.g. the memory space a piece of software uses, the actual footprint of an animal, the carbon footprint.

[3] Point set and not dot pattern as the method only uses the coordinates.

is obtained by using closed discs of radius $1/\alpha$ instead of half-planes; the α-shape is derived from this in a straightforward way. The authors do not discuss any principled way to choose the appropriate α for the type of shape required.

Chaudhuri et al. [3] present two methods for generating a footprint, called the *external shape*, from a dot pattern. Although they use the term 'dot pattern' they make no distinction between points and dots. For the first method, a grid of squares of side-length s is drawn on the plane, and the union of all grid-squares containing at least one of the dots is returned as the footprint, called the *s-shape*. For the r-shape they inscribe a disc of radius r round each dot, and draw an edge connecting any pair of dots whose discs intersect in a point not contained in any of the other discs. These edges provide an outline which, in our terms, may be regarded as the boundary of the footprint. As with the α-shape, no principles are given for selecting appropriate values of r or s.

Garai and Chaudhuri [10] propose a 'split and merge' method for generating footprints. This method starts from the convex hull and attempts to refine it to a shape more closely resembling what they refer to as the *underlying shape*. The method consists of three separate algorithms (four if the convex hull algorithm is included): *splitting*, *isolation*, and *merging*. This is one of the few algorithms that provides a way of *aiming* for a particular shape without having to re-run the algorithm with different parameters, so long as the user is able to identify a desired maximum area or number of sides just from a cursory examination of the dot pattern. Again the authors say little about the quality or type of footprint they generate.

Alani et al. [1] developed the *Dynamic Spatial Approximation Method* (DSAM). This system takes in both the dot pattern of the region to be found and the dot pattern of the area known to exist outside the region. It builds a Voronoi diagram based on these coordinates and takes the union of all the cells which contain an 'interior' point as its footprint. This work pays more attention than many in the area to the quality of footprint produced; this can be assessed in terms of how closely the region found fits the expected region. The existence of a contextually determined target shape differentiates this paper from others in the field.

Arampatzis et al. [2] follow on from Alani et al. [1]. However, they adapt DSAM to use Delaunay triangulations in conjunction with a system for finding point locations using web queries. They call this adaptation *the recolouring algorithm* and use it to generate boundaries for imprecise regions. Much like the DSAM this system has a target shape and, as such, this paper has more analysis of the footprint found than much of the field.

Galton and Duckham [9] propose two methods for finding footprints. The first method is a generalisation of the Jarvis March ('gift-wrapping') algorithm for convex hulls. The idea behind the Jarvis March is simple. From an origin point outside the dot set a radial half-line is swung in an arbitrary direction until it meets one of the dots. This dot is made the new origin point from which a radius is swung in the same direction as before until it meets another dot. This is repeated until the first dot is encountered again; the sequence of dots encountered in this way form the vertices of the convex hull. Dots are removed

from consideration if they have already been marked as being on the convex hull or if they lie within the area enclosed by the dots encountered so far. The 'Swinging Arm' algorithm is similar except that it uses a line-segment of some predetermined length instead of a half-line. The second method starts with the Delaunay triangulation and successively removes the longest external edge, subject to constraints of maintaining connectedness and regularity, until either some predetermined minimum length is reached, or no more edges can be removed. The authors note that there can be no uniquely 'optimal' footprint when the application context is considered to be general. The paper proposes nine criteria which may be used for evaluating footprint algorithms with respect to different application contexts, although little is said about any actual applications. Some of these criteria are used by the classification developed in the present paper.

Moreira and Santos [13] present a 'Concave Hull' algorithm. Like the Swinging Arm, Concave Hull is also derived from the Jarvis March algorithm, its difference being that it always selects the next vertex from the k nearest neighbours of the current vertex. This is the crux of the algorithm's effectiveness: by having a non-contextual integer as the variable that restrains the hull algorithm, they have a default base value from which they can run the algorithm (i.e. $k = 3$); if this fails to produce a footprint that satisfies the criteria (having no intersecting lines and containing all the points) then the algorithm is run with increasing values of k till such a footprint is created. Like most of the other authors they pay little attention to the quality of the footprint in relation to any application type, though they do mention the criteria given in [9]. Like the split and merge method [10], the Concave Hull algorithm requires some pre-processing of dots, using the Shared Nearest Neighbour (SNN) algorithm to determine any separable groupings in the dot pattern prior to running the algorithm. Like Garai and Chaudhuri they do not take account of this pre-processing algorithm in determining the computational complexity of their own.

Duckham et al. [4] provide a fuller account of the Delaunay-based method introduced in [9], now called the χ-algorithm. This paper includes a discussion of the footprint's properties, and how these are directly tied to the method by which it is created. More attention is paid to the choice of the length parameter l. There are practical limits on l for any triangulation (if it is too large then no lines will be removed, if it is too small too many will be removed) and consequently l can be normalised. Duckham *et al.* propose using this normalised parameter (λp) to find a starting value which should achieve what they call a *characteristic shape* for many, if not all, dot patterns. While they conclude that there is no λp that always produces a "good" characterization, the fact that they spend time considering this is unusual within the field. Unlike Moreira and Santos [13] and Garai and Chaudhuri [10], Duckham *et al.* do not discount the pre-processing (in this case computing the Delaunay triangulation and sorting the edges) when determining the complexity of the algorithm.

Galton [8], instead of proposing an algorithm, searches for objective criteria for evaluating the acceptability of any proposed footprint in relation to the 'perceived' shape of a dot pattern. The paper notes that in most of the published

work, "while lip-service is generally paid to the fact that there is no objective definition of such a 'perceived shape', little is said about how to verify this, or indeed, about exactly what it means". Restricting attention to footprints in the form of *polygonal hulls*, simple polygons having vertices selected from the dot pattern, all the other dots being within the interior, the paper presents evidence that while a dot pattern may have several equally acceptable perceived shapes, they all represent optimal or near-optimal compromises between the conflicting goals of simultaneously minimising both the area and the perimeter of the hull.

4 Classifications

4.1 Footprints

Before embarking on the classification, we make some preliminary observations of a general nature.

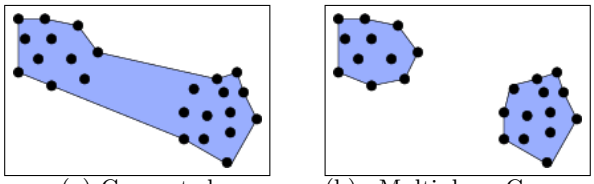
1. While a footprint is, considered in itself, a region with a shape, what makes it a footprint is the relationship it bears to the dot pattern from which it is derived. For this reason, it must be emphasised that our classification does not attempt to be a general classification of *shapes*; it only considers those aspects of shape which are relevant to the role of being a footprint.
2. Our classification criteria can be divided into *intrinsic* criteria, which concern properties of the footprint in itself, without reference to the dots, and *relational* criteria which concern the relationship between the footprint and the dots.

Intrinsic footprint criteria

[C] Connected *The footprint consists of a single connected component.*
Figure 1 shows examples of connected and disconnected footprints for the same dot pattern. Some algorithms will always generate a single connected component, implicitly assuming that any clustering has been done beforehand, with the algorithm being applied to individual clusters (e.g., Concave Hull [13], χ-shape [4]); others can yield footprints with multiple components (e.g., Swinging Arm [9]). The desirability or otherwise of multiple components is application-dependent, e.g., if only connected footprints are appropriate, use an algorithm guaranteed to produce such components.

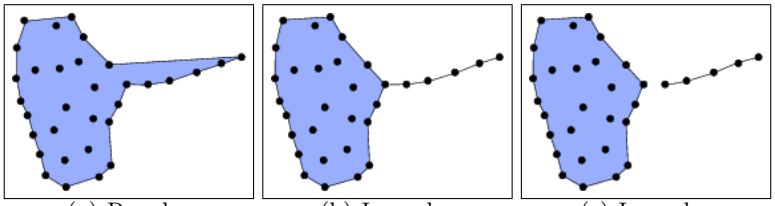
[R] Regular *The footprint is topologically regular.*
Assuming the footprint is topologically closed, this criterion amounts to whether or not the footprint contains boundary elements that do not bound the footprint's interior, such as the linear 'spike' in Figure 2(b) or isolated linear component in Figure 2(c).

[P] Polygonal *The boundary of the footprint is made up of only straight lines.*
For a polygonal footprint the boundary is made up entirely of straight line-segments as opposed to curves. (Figure 3).



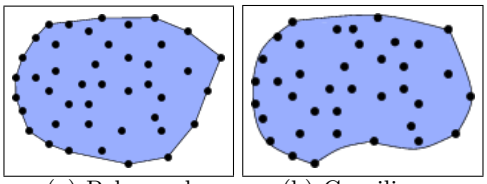
(a) Connected (b) Multiple Components

Fig. 1. Connectedness



(a) Regular (b) Irregular (c) Irregular

Fig. 2. Regular



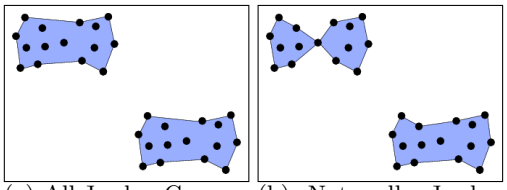
(a) Polygonal (b) Curvilinear

Fig. 3. Polygonal

[JC] Jordan Components *Each component of the footprint has a Jordan boundary.*

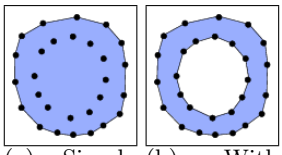
A Jordan boundary is a boundary which is a Jordan curve, i.e., homeomorphic to a circle. Such a boundary does not meet itself, so it is possible to traverse the entire boundary passing through each of its points only once. (Figure 4(a)). In Figure 4(b) the component with a non-Jordan boundary is represented as a 'bow tie' shape; of course this is not the only way the Jordan property can fail.[4]

[4] In relation to the 'bow-tie' configuration, if the footprint is formed by tracing out its boundary, then the constriction point may be either a *self-intersection*, where the boundary actually crosses itself, or a *pinch point*, where the boundary touches itself without crossing. An intersection or pinch-point may or may not occur on one of the dots; examination of the algorithms suggests that a self-intersection is more likely to occur away from a dot, whereas the opposite is true for a pinch point.



(a) All Jordan Components (b) Not all Jordan Components

Fig. 4. Jordan Boundary



(a) Simply Connected (b) With Cavity

Fig. 5. Simply Connected

[SCC] Simply Connected Components *Each component of the footprint is simply connected.*
A component that is not simply connected contains a 'hole' (Figure 5(b)). In two dimensions this means that the boundary is disconnected, with one of the boundary components facing the 'outside', and each other component bounding an internal cavity.[5]

Relational footprint criteria

[CED] Curvature Extrema At Dots *All curvature extrema of the footprint boundary coincide with dots.*
Very often a footprint is constructed by tracing its boundary through some or (more rarely) all the dots of the dot pattern. In such cases it is typical for the dots to mark curvature extrema of the outline; this is the normal situation when the outline is polygonal, with the dots at its vertices (Figure 6(a)), and is always found in the case of the convex hull.

Note that this criterion is independent of whether all, some, or none of the dots occur on the boundary (which is given by criteria [ADB] and [NDB] introduced next), as shown by Figure 6, where each value for one criterion can co-occur with each value of the other. However, [CED] $\wedge$ [NDB] (all curvature extrema are dots and all dots are off the boundary) can only be true if the foot-

[5] In three dimensions there are more varieties of connectivity to consider, e.g., the distinction between an internal cavity and a perforation. We shall not discuss these further here.

print is circular, in which case there are no curvature extrema, so [CED] is true by default.

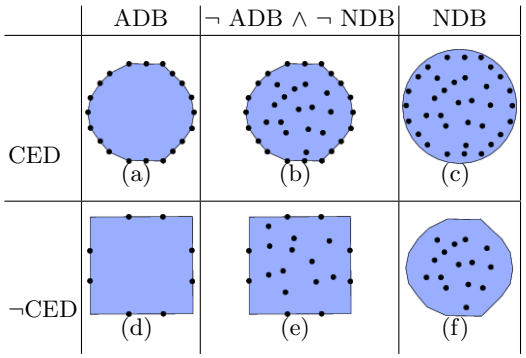


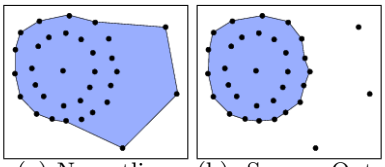
Fig. 6. Curvature Extrema and Dots On/Off Boundary

[ADB] All Dots on Boundary *All of the dots lie on the boundary of the footprint.*
In general we would not expect footprints to satisfy this criterion, but in some applications the dots are specifically intended to represent boundary points, and in such cases this criterion is appropriate. As mentioned above [ADB] is linked to, but distinct from, whether or not the curvature extrema coincide with dots (Figure 6).

[NDB] No Dots on Boundary *None of the dots lie on the boundary of the footprint.*
Criteria [ADB] and [NDB] cannot be simultaneously satisfied, and they are not independent. As with [ADB] it is linked to, but distinct from, whether or not the curvature extrema coincide with dots (Figure 6). Some algorithms (e.g., the Voronoi-based method of [1]) create footprints by amalgamating 'areas of influence' surrounding the dots. In such cases the dots typically all lie in the interior of the footprint, and hence off the boundary.

[FC] Full Coverage *All of the dots are included in the closure of the footprint.*
It is possible that a footprint algorithm may be able to distinguish certain dots from the pattern as 'noise', and as such it may wish to exclude them from the footprint. We call such dots *outliers* (Figure 7).

The criteria listed here can be combined together to give an overall classification of any particular footprint. For compactness, we shall represent such a classification using the abbreviations introduced above. For a footprint F we might present this in the form of a list such as



(a) No outliers (b) Some Outliers

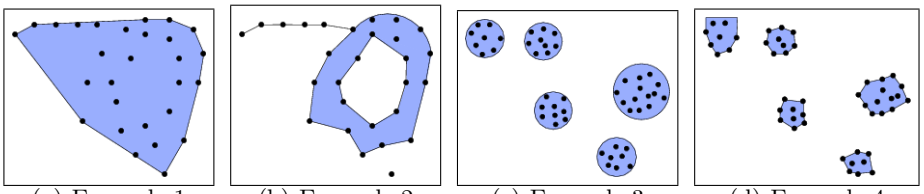
Fig. 7. Full Coverage

F{C, R, P, JC, SCC, CED, ADB, ¬NDB, FC}

but when comparing footprints it is convenient to present the data in tabular form as in Table 1, which presents the classifications for the footprints illustrated in Figure 8. In Table 1 the + indicates a true value and − a false.

Footprint Examples	C	R	P	JC	SCC	CED	ADB	NDB	FC
Example 1 [Figure 8(a)]	+	+	+	+	+	+	−	−	+
Example 2 [Figure 8(b)]	+	−	+	−	−	+	−	−	−
Example 3 [Figure 8(c)]	−	+	−	+	+	+	−	+	+
Example 4 [Figure 8(d)]	−	+	+	+	+	−	−	−	+

Table 1. Classification of Footprint Examples



(a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

Fig. 8. Footprint Examples

Note that Example 1 is the convex hull of its dot pattern, and the only properties for which it is not classified as + are [ADB] (all dots on the boundary) and [NDB] (no dots on the boundary). In general, the convex hull of any dot pattern will satisfy the criteria C, P, SCC, ¬NDB, and FC. Unless the dots are collinear, it will also satisfy R and JC. It may or may not satisfy ADB.

4.2 Algorithms

There are three types of algorithm classification criteria, relating to the nature of input expected, the process and the output. For the purposes of this paper we are most interested in the output produced, but we shall first briefly describe the other two types.

With regard to the input there are three key questions to ask: Does the algorithm require pre-processing? Does it require any data on the dots besides the coordinates (e.g. velocity or area)? Does it require an input parameter (e.g., line-length in Swinging Arm [9], α in α-shapes [5])?

The process has two criteria. First, does the algorithm build the footprint incrementally from the dot pattern or does it create the footprint by decrementally removing elements from some initial shape (usually the convex hull)? Second, does the algorithm optimise the footprint with respect to some pre-defined criteria designed to produce some form of best-fit shape (e.g., based on some general idea of what the final shape should look like). This optimisation can be performed in two ways. The first method is to produce a footprint, analyse it and then, if the footprint fails to match some pre-determined criteria, to iterate the algorithm with some internal value changed, as in Concave Hull [13]. The alternative is to examine the footprint to see if performing the next step of the algorithm invalidates some criteria, e.g., the χ-Hull algorithm [4] checks whether removing a line will make the footprint no longer a simple polygon. The two types may be called *remedial* and *preventive* optimisation respectively.

Output

This makes use of the footprint classification but with the added complication that the terms become modalised. There are many possible footprints for any dot set and infinitely many possible dot sets. We cannot predict all the footprints an algorithm can produce, but sometimes we can state whether a particular algorithm can ever produce a footprint with certain properties. Some properties are *necessary* (e.g., the output from Swinging Arm [9] is always polygonal), some are *unconstrained* (e.g., an α-shape [5] may have a cavity, but need not) and some are *impossible* (e.g., an s-shape [3] cannot have all its curvature extrema on dots). We can denote these cases using positive, null and negative respectively. Below, Table 2 shows the value system chosen to represent these.

Value	Description
1	All of the footprints produced by the algorithm satisfy the criterion
0	Some, but not all, of the footprints produced by the algorithm satisfy the criterion
−1	None of the footprints produced by the algorithm satisfy the criterion

Table 2. Possible Modal Values

Table 3 gives a few examples of algorithms as classified by the footprints they produce. As in the footprint examples we begin with the convex hull, using the Jarvis March algorithm. Any convex hull algorithm will be classified in the same way as they all produce the same footprint for a given dot pattern. Note that when assessing criteria R and JC, we discount cases where the input dots are all collinear, for which most algorithms will produce a straight line segment as the footprint. If in all other cases the footprint satisfies R or JC then, we shall record this using 1^- here (to mean 'all footprints for non-collinear inputs') rather than 0. Later, when using these values, 1^- will be treated as 1.

Algorithm Examples	C	R	P	JC	SCC	CED	ADB	NDB	FC
Jarvis March [11]	1	1^-	1	1^-	1	1	0	-1	1
Swinging Arm Algorithm [9]	0	1^-	1	1^-	1	1	0	-1	1
α-shape [5]	0	1^-	1	1^-	0	1	0	-1	0
Concave Hull [13]	1	1^-	1	1^-	1	1	0	-1	1
DSAM [1]	1	1	1	0	0	-1	-1	1	1

Table 3. Algorithm Examples

In Table 3 it can be seen that the outputs from the Concave Hull algorithm and the Jarvis March algorithm are classified in the same way. However, while they produce the same footprint types by the classification, they may be very different from the point of view of the quality evaluation criteria, as mentioned in §1.

4.3 Application Types

Little work has been done on relating the various methods to application types. It is understandable that there is a desire to abstract the theory away from the application, so that a general method for finding a footprint can be created and used in any circumstance. However, existing methods can produce very different results with a variety of different computational complexities. As such it seems that linking a method to, at least, a general application type would help the understanding of the aims behind each method and when it is best used. But before this can be done the application types have to be classified.

This is meant to be a broad description of the types of applications that these algorithms can be used for, not an in-depth study of each actual application. The classification is laid out with the idea that there are certain types of application and within each type there are optional requirements for each instance.

Pattern Recognition Finding a particular shape from an image is obviously linked to footprint generation, particularly in a digital field where the dots rep-

resent pixels. Within pattern recognition the relevance of associated data is apparent, so it is likely the algorithm will need data for the dots beyond their coordinates, e.g. colour values.

Classification In this type of application, the dot patterns do not exist in physical space, but represent positions in some abstract quality space, representing the attributes of different entities, each of which is assigned to some known class. The problem is to determine the region of quality space occupied by entities of that class. An example is shown in Figure 9, where instances of two different classes of entity (represented by the red and blue[6] colours) are shown. It is clear that there is a distinct difference between the groupings and a footprint algorithm might be used to find these. There is a presumption that the dots representing entities in the same class will form distinct groupings in the quality space — if this is not the case then that quality space has been inappropriately selected for the classification task.

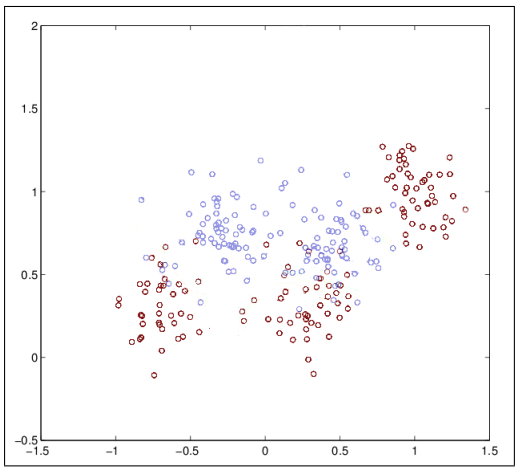


Fig. 9. Application Type: Classification (synthetic data from [6])

Region approximation Applications in this category are required to generate a geographical region on the basis of sample points within it. We distinguish three subcategories: simplification, estimation, and precisification. This represents a distinction in aims: the methods used in each case could well be the same.

Simplification occurs when the region itself is exactly known, but, perhaps for reasons of storage economy, the generation of a simpler version of its outline from a set of sample points within it is desired. An example might be that

[6] Darker and lighter if viewed in black and white

of approximating a county boundary from a collection of points known to lie inside it, as exemplified by the methods described in [2].

Estimation occurs when the region has a determinate boundary but this is incompletely known, e.g., if we try to locate the border of a city from the buildings within it. There is a definite boundary to the city but by using the footprint given by dot set (in which the dots represent a large number of the buildings) we might be able to approximate it.

Precisification Here there is no definite region to start with, for example the border of a forest. We have certain tree locations and we can use these to provide a sharp boundary to represent what may in reality be an indeterminate transition zone. The term 'precisification' is used in Supervaluation Theory [7], where it denotes the replacement of a category with incompletely determinate membership by one with fully determinate membership.

Aggregation This category is concerned with the creation of an aggregate entity from possibly scattered components, e.g., for the purpose of map generalisation, where what appears on a large scale map as a collection of individual buildings may need to be represented as a single connected built-up area when the scale is reduced. This is distinct from region approximation in that there need be no presumption that what is being represented is a region distinct from the buildings themselves; rather, what we want to represent *is* the buildings themselves, but at a coarse level of granularity where they cannot be distinguished as individuals.

With any of these categories one might wish to track changes over time as the entities represented by the dots appear, move, or disappear: in this case the application type becomes *dynamic* rather than *static*. This implicitly means we are treating our data as more than just coordinate locations as we have added a 'history' to each dot. The linkages between footprint types and application types may vary according as the latter are regarded as static or dynamic. However, for the scope of this paper we will restrict ourselves to the static form.

5 Classification Relations

As already mentioned, algorithm types are linked to the footprints they produce. Moreover, application types may be linked to the footprints types they require (§5.1).

5.1 Relating Application Types to Footprints

For a given application type, a particular footprint property may be required, permitted, or forbidden. For this reason we can use the same modal type classification as we used for describing the outputs from footprint algorithms (Table 2). This will also aid in choosing which algorithms can be used for a specific application type. The tables below give our initial thoughts concerning the footprint requirement of the various application types; more detailed analysis will be required for a more definitive statement of these, as described below in §6.

Application Type	C	R	P	JC	SCC	CED	ADB	NDB	FC
Pattern Rec.	0	0	0	0	0	0	0	0	1
Classification	1	1	0	1	0	−1	−1	1	1
Simplification	1	1	0	1	0	0	0	−1	0
Estimation	0	1	0	0	0	0	0	0	1
Precisification	0	1	0	0	0	−1	−1	1	0
Aggregation	1	1	0	1	0	0	0	0	1

Table 4. Application Types Related to Footprints

5.2 Application Types to Algorithms

Just as an application type can have requirements on the footprints, it may also be constrained as to what data it can provide to an algorithm. However, this kind of constraint may not be relevant at the level of generality with which we are treating application types here.

There is one algorithm criterion which is of particular interest. Whether or not the dot pattern contains only coordinate data is likely to be heavily affected by making the data dynamic. When running whichever loop is required to allow the algorithm to generate the footprint over the 'frames' that make up the data (live or pre-set) it would be naïve not to keep track of the 'past' associated with each point. Without this the main body of the algorithm needs to be re-run for each frame, instead of using the knowledge of the current footprint and likelihood of movement for each dot to cut down on processing time.

Algorithm Suitability By comparing the classification of an algorithm with the application-type classification, we can endeavour to determine the suitability of different algorithms for various applications. Earlier we introduced a value system for our modalities. The footprint criteria were given a value of 1 if it was necessarily true that it holds for all elements, −1 if it was necessarily true that it does not hold for any elements and 0 if the criterion was unconstrained. Application contexts have a vector of these values for their required footprints and the algorithms have vectors for their possible output footprints.

With the goal of having a clear, systematic and quantitative approach we looked at several ways of treating the results. The first method that presented itself was to treat these vectors as points in nine dimensional space, allowing us to find the Euclidean distance between them and thereby assess, numerically, the suitability of the algorithm. As the maximum possible distance between any of the values is 2 we can have a maximum total distance of $\sqrt{9 \cdot 2^2} = 6$. If the two vectors are equal we have the minimum distance of 0. This allows us to present the 'quality' of the algorithm, with regards to the application, on a scale of 0–1, where 1 is best fit and 0 is worst, using the formula $1 - (x/6)$ where x is the distance between the two classification vectors.

However this assumes that all the footprint criteria should be treated as of equal weight. It may well be the case that an algorithm is shown to be close to a best fit but that the criterion it fails is the most important for a specific context. There is also the issue that vectors with many small single value differences are the same distance apart as vectors with few large differences (four 1 value differences to each 2 value). This goes some way to depicting the importance large value differences have over small ones and as a result may be of benefit, but the degree of increase in worth is simply a product of the way the distance is found and therefore may be inappropriate.

Despite these concerns the 'real' issue with the Euclidean distance is how to interpret the result. When the method was used on the previously mentioned algorithms against the application vectors the results were all in the range of 0.38 to 0.76, with most of the values being very close for each application. These close results made it difficult to determine which algorithms would perform better than others, particularly as assigning the qualitative value of 'good' to a strictly quantitative measure seemed presumptious, even pseudoscientific.

Further to the Euclidean distance we looked at using the scalar product and the angle between the vectors, but both these methods suffered from the same problems as the distance.

The facts that the value system indicates that all the criteria are equally weighted and that the resultant values are so similar are clear indicators that the three values may not be enough. One approach to correcting this would be to relate applications to footprints using a continuous scale from -1 to 1, allowing the values to be ranked by importance. This could also be applied to the algorithms: on the original discrete scale an algorithm which is connected in all but the rarest cases would still be marked as a 0, whereas it may be more appropriate to assign a value of, for example, 0.9, indicating that the algorithm mostly produces connected footprints. This continuous scale could also lead to greater differences between the distances so assessing them would be easier. However there is still the problem of judging the result in a systematic manner and the added problem of assigning the original values, for example how do you judge if an algorithm is '0.45' on a scale of producing a regular footprint? Even if you take it as a likelihood for how often an algorithm produces a type of footprint then you have an issue, if an algorithm only produces an irregular footprint when the dots are collinear there are still infinitely many dot patterns which will cause this.

The duality in meaning of 0 is also worth noting. The value 0 can mean two separate things for an application: (1) the algorithm is required to be able to produce both extremes; or (2) the algorithm does not care which value is produced. Thus if the application vector has 0 indicating 'uncaring' in a certain position, this should be allowed as a good match with any of 1, 0, and -1 in the algorithm vector, whereas a 0 indicating a requirement of both values should only match to a 0 on the algorithm.

Seeing that there were more than a few issues with the system it seemed sensible to go over some specific application fields and attempt to find their

appropriate vectors. In doing this we hoped that the answers to the above issues would present themselves. However after just a few it became apparent that even an application as specific as 'removing outliers from a spatial distribution' gave a vector composed almost entirely of 0's, it often being the case that a dot pattern can be envisioned for each possible value for a criterion. These 'weak' results would indicate that the dot pattern heavily influences the requirements for the application.

Based on the influence the dots have it appears that an exact method for assessing the suitability of the algorithm requires a change in the classification approach. Instead of simply stating that the application always/sometimes/never requires footprints of a given type as output, we might now additionally try to characterise the inputs for which outputs of that type are to be produced, thereby moving towards providing a potential specification for an algorithm satisfying that application. A similar system could be applied to the algorithm, by specifying the types of output associated with different types of input. These two classifications could then be compared to check for suitability. Unfortunately the process for creating such a system is not known. It would require having a complete set of dot pattern classifiers and a way of assigning values to them. Then the system would have to be created by considering which of the dot pattern descriptors affect which footprint classifiers and in what manner. For example; if the minimum and maximum distances between any two points differ largely from the average distances does this affect the likelihood that the footprint will need to be connected? This would also need to be done for the algorithm in terms of how the dot pattern affects the footprint the algorithm produces. It is likely that such a process would be complicated and prone to assigning values as arbitrarily as in the suggestion for continuous values.

After much work it became apparent that we needed to 'tighten' our focus. Our goal could be achieved much more simply and straightforwardly using direct comparison. This still leaves us with the problem of the dual meaning of 0 for the applications, and the fact that an algorithm which will produce one type of footprint except in special circumstances (i.e. collinearity) will strictly be given a value of 0. As such we will re-introduce the 1^- and, if necessary, add -1^+ to both algorithms and applications; these indicate that except in special circumstances, which should be described, the algorithm produces, or the application requires, a value of 1 or -1 respectively. We will also strictly define the 0 on an application as requiring both, if the application has no preference for the result the value is a $\sim$ sign. These extended values allow for very easy comparison, users can concern themselves only with values about which they care. They can also clearly see which special circumstances can occur and decide if they are applicable to their application. An example of the way this could be set out is shown in Table 5.[7] Running through this we can see that the Swinging Arm algorithm [9] fails completely on SCC and FC and only satisfies R and JC if the special cases are

[7] For the sake of the example we have made the assumption that, for this application, curvature extrema are required at dots even though this is not stated by the application title.

likely to come up regularly in the application field. As such we can say that the Swinging Arm algorithm would be unsuitable for this application.

	C	R	P	JC	SCC	CED	ADB	NDB	FC
Algorithm:	0	1^-	1	1^-	1	1	0	-1	1
Application:	0	0	$\sim$	0	0	1	0	-1	-1

Table 5. Assesing Swinging Arm [9] for suitability to the application of removing outliers from a spatial distribution. Special cases on the algorithm: R and JC are -1 when dots are collinear.

6 Conclusion

Before discussing the additions this work has made to the field the shortcomings must be explained. The values given on the application types are easily debatable, and perhaps some of the values could even be changed to their polar opposite while keeping in line with the application descriptions. However, we chose the values while attempting to keep in mind the aim of maintaining a very general view. Even should there be any significant disagreement with the values the goal of this paper is unaffected, our aim being to show the possibility of developing a systematic value system that can be used to rate algorithms against applications.

We noted in §3 that current literature has little to say about the types of footprint generated or required. We advocate the use of a classification system along the lines presented here, though we expect that further work will refine the details considerably. There is also a noticeable deficiency with regard to applying the algorithms to any applications. Even from the very general definitions given in this paper it should be apparent that the contexts can differ hugely and may have conflicting requirements. Given this diversity of applications, it seems strange to present a footprint algorithm without linking it to an application context, without which little can be said about the suitability of the footprints generated. The disparity also means a truly general algorithm is unlikely. The algorithms considered here were largely produced without specific applications in mind; it is interesting to speculate how different they might have been had they been created with particular contexts in mind.

Aside from the above mentioned shortfalls the paper presents a systematic way of rating the appropriateness of algorithms for applications by classifying the footprints they create. This rating uses a clear nomenclature which is easily repeatable and therefore usable by others. In further work we plan to refine the list of criteria, to examine the dot pattern types more closely, and to consider how input and process criteria for the algorithms relate to application types.

Bibliography

[1] H. Alani, C. B. Jones, and D. Tudhope. Voronoi-based region approximation for geographical information retrieval with gazetteers. *International Journal of Geographical Information Science*, 15(4):287–306, 2001.

[2] Avi Arampatzis, Marc van Kreveld, Iris Reinacher, Christopher B. Jones, Subodh Vaid, Paul Clough, Hideo Joho, and Mark Sanderson. Web-based delineation of imprecise regions. In *Computers, Environment and Urban Systems*, volume 30, pages 436–459. Elsevier, 2006.

[3] A. Ray Chaudhuri, B. B. Chaudhuri, and S. K. Parui. A novel approach to computation of the shape of a dot pattern and extraction of its perceptual border. In *Computer Vision and Image Understanding*, volume 68, pages 257–275. Academic Press, 1997.

[4] Matt Duckham, Lars Kulik, Mike Worboys, and Antony Galton. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. In *Pattern Recognition*, volume 41, pages 3224–3236. Elsevier, 2008.

[5] Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. In *Computer Vision and Image Understanding*, volume IT-29, pages 551–559. IEEE, 1983.

[6] J.E. Fieldsend, T.C. Bailey, R.M. Everson, W.J. Krzanowski, D. Partridge, and V. Schetinin. Bayesian inductively learned modules for safety critical systems. In *Computing Science and Statistics*, volume 35, pages 110–125. 2003.

[7] Kit Fine. Vagueness, truth and logic. *Synthese*, 30(3-4):265–300, September 1975.

[8] Antony Galton. Pareto-optimality of cognitively preferred polygonal hulls for dot patterns. In *Spatial Cognition*, 2008.

[9] Antony Galton and Matt Duckham. What is the region occupied by a set of points? In *GIScience*, 2006.

[10] Gautam Garai and B. B. Chaudhuri. A split and merge procedure for polygonal border detection of dot pattern. In *Image and Vision Computing*, volume 17, pages 75–82. Elsevier, 1999.

[11] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. In *Information Processing Letters*, volume 2, pages 18–21. North-Holland Publishing Company, 1973.

[12] R Klette and A Rosenfeld. *Digital Geometry: Geometric Methods for Digital Picture Analysis.* Morgan Kaufmann, 2004.

[13] Adriano Moreira and Maribel Yasmina Santos. Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points. In *International Conference on Computer Graphics Theory and Applications GRAPP*, 2007.