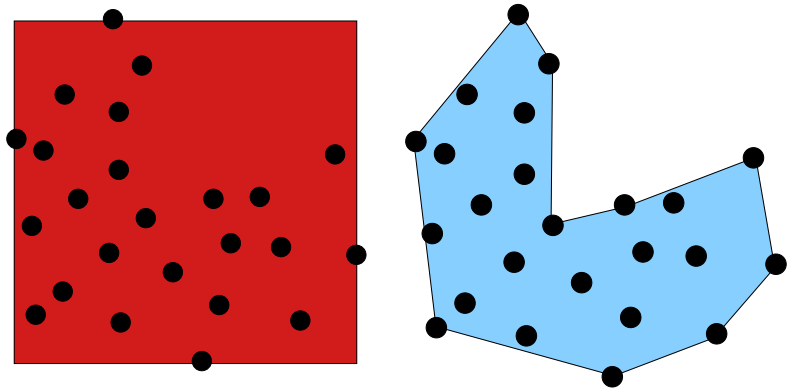
1 Introduction

1.1 Overview

Many phenomena, real and abstract, can be represented by sets of point locations. These *dot patterns* occur in many fields including Geographic Information Systems (GISs), classification and optimisation. Dot patterns are a useful abstraction allowing for many types of mathematical analysis to be performed e.g., using statistical analysis methods to see if the spread of leukemia within the vicinity of a nuclear power station is significant. Sullivan and Unwin [51, ch. 5.1] detail such analysis while commenting on the 1984 report by Sir Donald Black [12].

Often a way of approximating the region that contains a pattern is required, and an areal or volumetric object used for this approximation can be called a *footprint* of the dot pattern. Footprints can be used to reduce the memory space taken up by the pattern, and in some cases they can be interpreted in such a way as to increase the known information about the phenomenon underlying the pattern. For example, consider the case of a location aware application that wishes to identify the boundaries of a city by the locations of buildings sampled from within it ([50, 2]); the boundary locations are not present in the data but can be approximated by a footprint.

In theory any bounded region is a candidate footprint for a given dot pattern, however it is intuitively obvious that some footprints are better than others. Just how appropriate a footprint is for a dot pattern is a clearly context specific problem. However there are some general distinctions that can be made. One of the goals of footprint creation is to make clear the information that the pattern represents; a footprint that only serves to obfuscate



(a) Minimum Isothetic Bounding Box (b) Non-convex Footprint

Figure 1.1 Footprint as a Representative

is unlikely to be desirable. In Fig. 1.1 we can see two footprints (Fig. 1.1(a) and Fig. 1.1(b)) for the same dot pattern. Fig. 1.1(a) may be a sufficiently 'good' representation for the application context but Fig. 1.1(b) provides more information about the spread of the pattern. In using clarity of salient information as a criterion we can state that Fig. 1.1(b) is a 'better' representation than Fig. 1.1(a) of the dot pattern. We must be aware, however, that there is generally an inverse relation between the information content of the footprint and its computational complexity; this trade-off is discussed in greater detail in Chapter 4. Current work in the field of footprint generation has not focused strongly on general methods for assessing the quality of a footprint. Most algorithm authors provide a set of requirements on the types of shape they allow (e.g., no degenerate lines) but judge the quality of the footprint on human intuition alone. Galton [27] presents an exploratory set of findings on a study designed to find the mathematical properties which lead to one footprint being intuitively 'better' than another. The results show a strong tendency for people to attempt to optimise the trade-off between minimising the area and minimising the perimeter, but Galton also posits that there may be other factors at play such as sinuosity and cultural cues (e.g., similarity to alphabet characters). Dupenois and Galton [21] present a classification of footprints that is intended to be used to delineate algorithms by the footprint types they produce. The classification does not itself make statements on footprint quality but by delineating the footprints the user can choose the algorithm most likely to produce the appropriate footprint types for their application. An extension and discussion of this work is presented in Chapter 4.

The convex hull[1] has existed as a mathematical construction for many years. However one of the earliest papers in which it is used explicitly to find an appropriate region for a point set was by Jarvis in 1973 [39]. While not the most efficient algorithm, Jarvis' paper identified the need for such work in the field of pattern recognition. Ten years after Jarvis' work the much cited Edelsbrunner *et al.* [23] presented the α-shape as one of the first region approximations of a point set that could produce concavities. 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* 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. Edelsbrunner *et al.* were also working in the field of pattern recognition, one of their given examples being character recognition from sampled images. Footprints have expanded in use across several fields, being used in image processing (e.g., Edelsbrunner *et al.* [23, 24, 22]), pattern recognition (e.g., Gofman [29]) and GIS (e.g., Alani *et al.* [2]). The prolific nature of footprint use has given rise to a large, and increasing, number of different algorithms approaching the problem from a variety of viewpoints: The χ-hull [20], for example, uses the Delaunay triangulation of the dot pattern and successively removes edges longer than a threshold length[2], whereas the k-nearest neighbours algorithm [50] builds a footprint by, from some origin dot, iteratively selecting the next dot on the hull from the set of k nearest dots. The two algorithms appear to operate from different base concepts, using 'destructive' and 'constructive' viewpoints respectively.

[1] Unless otherwise specified we use convex hull to mean minimum convex hull.

[2] Assuming removal does not invalidate any of the hull requirements stated by Duckham *et al.* .

There is a distinction to be made between the concepts of a hull and a footprint. Footprint is a general term for any candidate region that has been assigned to a dot pattern. Whereas hull is more specific, Klette and Rosenfeld [42] define a hull with three requirements, given a hull operator H and set of subsets $\mathbf{S}$ [3]:

[**H1**] $\forall M \in \mathbf{S}\ M \subseteq H(M)$

[**H2**] $\forall M_1, M_2 \in \mathbf{S}\ M_1 \subseteq M_2$ implies $H(M_1) \subseteq H(M_2)$

[**H3**] $\forall M \in \mathbf{S}\ H(H(M)) \subseteq H(M)$

Klette and Rosenfeld note that [**H1**] and [**H3**] imply $H(H(M)) = H(M)$ but they do not replace [**H3**] with this implication as they define variations of the hull (pseudohull and near-hull) that have differing combinations of these requirements (along with a fourth). Many of the footprint algorithms in the literature use hull as part of their naming convention (convex-hull, α-hull, χ-hull, etc.) and not all fit with Klette and Rosenfeld's strict definition (for example the α-hull need not contain all the dots of a pattern). Further, there tends to be an understanding of a hull as being minimal in some respect. A footprint can extend beyond the the dot pattern, with no dot coinciding with its boundary. Although we note that, in general, the more useful footprints tend to be minimal because, as was mentioned earlier, it is wise to avoid footprints that only serve to make the information the application is interested in less clear. For the purposes of this thesis, and to avoid confusion in general, we will take all hulls as footprints but only use hull to describe footprints created by the few algorithms that already use it as part of their naming convention.

The range of different algorithms that can produce footprints that are not the convex hull tend to have some control parameter. Change in this parameter often leads to change in the 'spikiness' of the footprint; varying the concavity it presents. The parameter is a necessity because the patterns can vary so greatly that no method without such a parameter can be said to always produce an 'appropriate' footprint. With a control parameter the algorithms can be adjusted until their output fits the context specific definition of correctness. This parameterisation often leads to problems when using the algorithms. For many methods the parameter is an abstract measurement; the best value for which is not immediately obvious to a human user. The α-hull, for example, uses $1/\alpha$ as a radius for discs as part of its process; for a human to give a value of α that will produce a context appropriate footprint is largely trial and error. Even the algorithms with more apparent parameters (e.g., length of edge to remove in χ-hull) still often require some adjustment to find the best region the algorithm can produce for the context. When the set of dots is no longer static there is the added complexity that a parameter value that worked well at one timestep may no longer provide a good fit region at a later time-step. Parameterisation is an important factor when discussing footprint algorithms and in the examination of the change identifiers this thesis introduces, consequently it will be revisited in the chapters dealing with these topics (Chapters 4 and 5 respectively).

[3] where each element of $\mathbf{S}$ is a set of points

While the footprints for static phenomena have been the centre of much research there has been far less inquiry into how a region may be maintained over a changing collective phenomenon (its members can move, be added or be removed). Examples in which the phenomenon is subject to change are ubiquitous and cover a range of fields, e.g., tracking animals ([46]); identifying ship movements, to avoid collisions and traffic; understanding the behaviour of crowds in shopping centres ([3]); and tracking the populations of optimisation problems across multiple generations (this will be expanded upon in Chapters 8 and 10). Additional to these is the emergent field of using sensor networks to provide real-time updates in emergency systems about the state of a current situation, for example the spread of toxic gases, wildfires and floods ([40, 41]).

Existing dynamic footprint work has looked primarily at using the convex hull (e.g., [52, 15, 6, 36, 33, 34]). The convex hull is a strictly defined mathematical construct and is uniquely defined for any single pattern. Footprints considered more generally do not have this unique definition on a pattern. Neither do they often have a short, simple mathematical definition; their construction being the product of their algorithm and can not be reduced into a single statement. The non-uniqueness of the algorithms arises from the parameter (as discussed above) that most algorithms have for controlling some aspect of their formation. Current work uses the strong mathematical properties of the convex hull (e.g.,[15, 6, 33, 34]) to create 'certificates' that can be checked at each time step for failure. A certificate is a small, easily-computable property of the footprints relation to the dots. In the event of a failure the footprint is updated, either locally at the point of failure or globally. Footprints in general can not have certificates in the same fashion as there are no definite properties to check[4]. However, fortuitously this vagueness allows us to state that if a footprint is a suitable representation at a timestep t_n, under most conditions of change it will still be appropriate at t_{n+1}. Instead of checking the footprint in relation to the dots for suitability we can examine the pattern itself to see if sufficient change has occurred such that the footprint must be updated. This checking requires a method of measuring change on dot patterns and a way to assign an appropriate change threshold.

There is a body of work that concerns itself with describing change for spatial-temporal entities like dot patterns and this is examined in Chapter 2. The intial difference in our approach to the existing research is in the definition we give the dot patterns. By taking an individual pattern as a mathematical static abstraction its properties can be examined without the complication of change. The differences between these measurements for two different patterns can be used as measures of how much change needs to occur in each property for one pattern to match the other. Taken across a wide range of properties these measurements provide a value for the total change a pattern must undergo to be equivalent to another. Previous work has looked at dot patterns as part of a continuum over which change must be measured. Both approaches lead to similar measurements (for example both are likely to lead to a measure of change in location), however by looking at the static properties first we are able to use measurements from different fields such as

[4]When such properties exist they are prescribed by the context and as such can not be relied upon to exist in all contexts, for example an application that requires all dots exist within the footprint.

statistical analysis and are able to draw distinctions between different measurement types (see Chapter 5).

1.2 Terminology

The sheer scale of work that uses constructs similar in nature to dot patterns means that there is a high probability of confusion when comparing work by different authors. Before we present the definition of dot patterns that is used for this thesis it is important to note that not all existing work treats dot patterns in the same way and we will clarify the differences when they are relevant.

Dot

A *dot* is an ⟨id, location⟩ pair. It is a representative data point of any phenomenom that can be assigned a location within a space (whether real-world or abstract). The pairwise nature of a *dot* renders it very simple, since it is only when they are grouped and moving does complexity arise. The term dot, over the more common *point*, is used for two reasons. Firstly, a dot may have more data associated with it than just a location, for the definition used in this thesis dots have an identifier. Secondly, to draw a distinction between the dot as a member of a dot pattern and a point within the dot pattern, for example the mean center (centroid) of a pattern is a point that may not coincide with a dot.

As an addendum to the definition of the dot it should be noted that the work presented in this thesis does not make use of the identity attribute; this is to reduce the number of assumptions made about the raw data and will be discussed in Chapter 3. However there is no reason that descriptors, and change identifiers, that use the dot identity can not be created for applications which can guarantee its availability.

Dot Pattern

A set of dots is called a *dot pattern*. It is fully exhausted by these dots and is a mathematical abstract. Its mathematical nature means it is incapable of change.

Dynamic Dot Pattern

As a dot pattern cannot undergo change we need a structure that uses dot patterns to represent the change of the underlying phenomenon. The *dynamic dot pattern* is a function mapping from time to dot pattern, as shown in Fig. 1.2. The function maps such that $\forall\ \tau \in T$, $DP(\tau)$ is the dot pattern representative of the underlying phenomenon at time τ. Rather than use $DP(\tau)$ to refer to a dot pattern at a specific time we use the nomenclature *phase* (for which we borrow the wave notation of ϕ). For example the pattern in Fig. 1.2

at timestep τ_0 is the first phase (ϕ_0) of the dynamic dot pattern and the pattern at τ_1 is the second phase (ϕ_1).

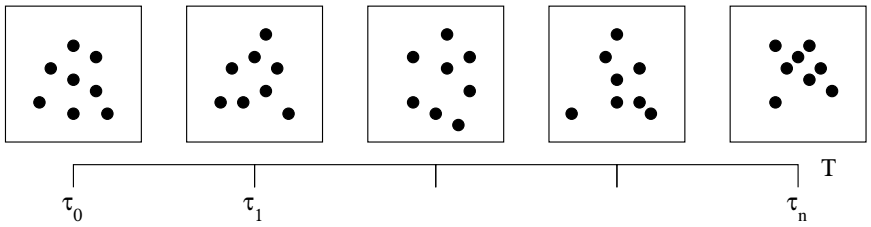


Figure 1.2 Time Domain (T)

The phenomenon that the dynamic dot pattern represents is likely to be changing in a continuous fashion, however the phases are discrete steps. Thus the timesteps are used to provide a discretised time domain for the dynamic dot pattern to map from. There will almost certainly be change occuring between the timesteps, and therefore between the phases, but for the purposes of the work presented in this thesis we can ignore the 'in-between' dot patterns as we are assuming that the *granularity* of the arriving data is appropriate to the application.

Collective

A *collective* is a grouping of objects with some attribute in common[5]. A dot pattern is representative of the locations of the members of a collective at an instant in time. Hence, a collective may have a continuant identity whereas a dot pattern's identity is given entirely by the locations of its dots and its time value. This means that a dynamic dot pattern for which each phase comes from the same collective is representative of the change the collective undergoes over the given time domain.

Descriptor

Dot patterns have various mathematical properties that arise from the cardinality and the locations of their member dots; for example the standard deviation from the mean centre. There is an incalculable number of these dot pattern *descriptors* but many will be measuring similar attributes of a pattern in different ways (e.g., bounding box and standard deviation can both be said to be measures of the extent of the pattern). We propose several *classes* of descriptor that are general headings for types of information that can be retrieved from the pattern. Further to the mathematical descriptors these classes can provide qualitative information (e.g., a pattern is large) and such information may be advantageous to a user, particularly in application contexts where response time is important. There are problems faced in using these qualitative statements, primarily as they tend to be context specific, but they do not affect the central preposition of this work so discussion on qualitative assertions are left until the chapter concerning future work.

[5]Rather trivially (and tautologically) this attribute can be the fact that they have been grouped together

Change Identifier

The entities that are represented by individual dots in a dot pattern can only change in a three ways: appearance, disappearance and translation[6] ([58, 38]). The dot pattern phases within a dynamic dot pattern can, therefore, only differ in the location and the cardinality of their members. The differences can be measured using the descriptors of the dot patterns, and comparison of the extent of these differences is a measure of the change undergone between the phases. Simple entity changes can lead to complex pattern differences and, if change is to be measured in any formal sense, we must be able to measure the difference in such a way that none of the emergent complex behaviours are ignored. Change within a descriptor class represents an emergent complex behaviour from the collective. For example:

> Two phases differ in standard deviation in that the deviation at the later time step is greater than in the earlier. Standard deviation is within the class of extent descriptors. We can therefore infer that the collective has undergone the complex behaviour of expansion.

Further to the change identifiers based on descriptors, there are standard metrics which may be used to measure the difference dot patterns. For example, the distance between the mean centres of two patterns.

1.3 Summary of Thesis

And the Novelties it Presents

The background chapter presents an overview of the existing literature from the fields that frame this thesis. It begins with a brief study on how dot patterns have been represented in traditional Geographic Information Systems (GISs), including an examination of some of the possible data structures that may be used to contain the patterns. The chapter continues with discussions of the literature for the fields of spatio-temporal data, shape description, footprints and footprint algorithms, and dynamic region maintenance.

After the literature review given in Chapter 2 further background is provided by the next two chapters, which discuss dot patterns and footprints (Chapters 3 and 4 respectively). The dot patterns represent the input and footprints the output of the change identifiers, and before discussing change identifiers themselves it is necessary to understand the structures they use. Underpinning the change identifiers is the novel examination of the dot patterns as mathematical abstracts removed from the collective that they represent,by which the mathematical properties (descriptors) of the patterns which describe them can be isolated. Chapter 3 presents a detailed discussion of these descriptors and how the ones used within this thesis were decided upon. The descriptors provide a definition of a dot pattern that allows us to examine the myriad types of pattern that exist and, more

[6] The entities could undergo many other types of change, ageing for example, but for our abstraction only the three given apply.

importantly, compare them. It is this comparison which leads to the concept of change identifiers and the generalised framework in which we use them to measure change across a dynamic dot pattern. The footprints chapter presents a classification of footprint types (first published in [21]) with a discussion on how the classification relates to dynamic dot patterns. The classification is used to show that the footprint algorithms, which tend to be created for generalised fields rather than specific applications, can be paired with applications that best suit the footprint types they produce.

Chapter 5 builds on the foundations laid down by the previous chapters, particularly that of Chapter 3, to formally define the change identifiers; distinguishing between two different types of identifier: one that uses dot pattern descriptors and another that makes use of standard difference measures (metrics). The chapter presents the core novelty of the thesis; detailing how the change identifiers can be used as a method by which to measure change in a collective of spatio-temporal entities represented by a dynamic dot pattern. The use of multiple change identifiers is explored within this chapter and methods to combine their measures fairly are discussed. Finally, the change identifiers chapter provides a method for assessing the quality of a change identifier and a change identifier set using the trade-off between time saved and how well the footprint is tracked.

Having given detailed descriptions of the input, output and process components (dot patterns, footprints and change identifiers respectively) of the change identifier framework, Chapter 6 (the methodolgy chapter) presents the system that combines them. The chapter demonstrates the modular nature of the framework used for the results in this thesis and how it can be employed for testing the identifiers and for real-world applications. The methodology also describes the data structure used for storing the dot patterns and the reasoning behind the choice of format that the dot patterns are expected to arrive in. The chapter finishes with an examination of some of the ways in which the difference between footprints can be measured, as such measures are important to the way in which the quality of the change identifiers can be assessed.

Since the results span a large number of experiments there are a number of different graphs produced, consequently the results are separated into their own chapter (Chapter 7). The chapter also provides an explanation for the selection of the dynamic dot patterns and footprint algorithms that are used within the experiments. This differs from the discussion in Chapter 6, which described the way in which the framework is constructed and the methods by which to use it, because the results chapter details the sources used within the experimentation.

Chapter 8 confronts the problem, that faces any user of change identifiers, of selecting which identifiers are most appropriate. This chapter makes use of the Evolutionary Algorithm tools from the optimisation field of research as a way of assessing the change identifier sets against each other, which, as far as this author is aware, is the first time Evolutionary Algorithms have been applied to quality assessment within the field of measuring spatio-temporal data.

The conclusions chapter re-iterates the areas in which the change identifiers have performed

as expected, as well as those areas in which they can be improved. The conclusions drawn lead to the final chapter (Chapter 10) of this thesis, which concerns itself with the possible ways in which the change identifiers can be taken forward. The future work chapter begins with an examination on how dot pattern descriptors may be used to define different types of dot pattern and shows some preliminary results using an agglomerative clustering technique to partition the dot pattern classes. The dot pattern classification is followed by a consideration of the use of change identifiers to indicate when the footprint type has changed according to the classification given in Chapter 4. Further to the discussion of footprint types the chapter proposes using change identifiers to dynamically alter the footprint algorithm parameters based on the change the collective is undergoing (as represented by the dynamic dot pattern). Finally the chapter, and the thesis, finishes with some ruminations about where change identifiers may be used and whether or not they could return salient qualitative information.

2 Background

This thesis is situated within, and contributes, to the discussion of two complementary fields. The first is the analysis of spatio-temporal data and the second is the assignment of regions (footprints) to point-based data sets. There are large bodies of work for each and the intersection of these fields has been used in Geographic Information Systems (GISs [1]) as a form of query filtering (e.g., DSAM [5]). Most of this work has dealt with data sets that are solely spatial or in which change is glacial in pace (regions of forests, cities, etc.). This thesis focuses on forming a general framework with which to maintain footprints across spatio-temporal data.

To situate this work in the associated research we will provide a quick overview of the nature of dot patterns when considered within GISc and the strucures that may be used to contain then, after which approaches to the classification and study of spatio-temporal data will be examined. This will be followed by a general overview of shape descriptions and footprint algorithms. Finally the existing research on dynamic tracking will be looked at. This program of study will provide the necessary background required to further discuss the themes of this thesis.

2.1 GIS and Dot Patterns

Sullivan and Unwin [51, ch. 4] give a good description of the treatement of dot patterns (called point patterns) from a geographic standpoint. The chapter begins by noting that point patterns frequently occur in GISs and gives the examples of crime or death hot-spot analysis. The members of a point pattern are termed *events*, and each event represents a single object of interest from the region being studied. Sullivan and Unwin also state that a set of events is only a point pattern if it conforms to a number of criteria:

---

1. The pattern should be *mapped on the plane*.
2. The study area should be *determined objectively*.
3. The pattern should be an enumeration or *census* of the entities of interest, not a sample.

[1] Occasionally there is some confusion as to whether GIS should stand for Geographic Information System or Geographic Information Science. Within this thesis we will be using GIS when we refer to a system and GISc for the science as a whole.

4. There should be *one-to-one correspondence* between objects in the study area and events in the pattern.

5. Event locations must be *proper*. They should not be, for example the centroids of areal units chosen as representative ... They really should represent the point locations of entities that can be sensibly be considered points at the scale of the study.

Sullivan and Unwin [51, ch. 4]

Of the requirements only requirement 2 can be generally applied to dot patterns; the wide array of fields in which dot patterns are used means that the other requirements are not always satisfied. For example, requirement 4 assumes a direct mapping between dots and a real-world object and 3 states that a the pattern should not be a sample, however a dot pattern may represent a sampling from a region [2] or a set of classification results. The requirements are given despite the differences between dot patterns and point patterns because much of the spatio-temporal literature comes from the GISc field, and we should be aware of the assumptions that the work makes.

Sullivan and Unwin examine two approaches to point pattern analysis: point density and point separation. Density measures can be used to show first-order effects while separation is indicative of second-order effects. A first-order effect occurs when the physical location has correlation with the event, for example in a study of the locations of the swans in Hyde Park it is likely that the clustering would occur around the bodies of water. A second-order effect occurs where an event affects the incidence of other events, for example a study of locations of a particular contagious disease would have areas of clustered points as the probability of catching the disease increases with the number of events in the area. The static way in which we examine dot patterns will not be able to draw a distinction between the two types, however the change identifiers may be split along the same delineation. This suggests that a possible avenue for future research is the classification of types of change identifiers and how they can be used to inform a user of the more complex behaviours that the collective is exhibiting; i.e. Is a collective's change representative of a first or second order effect?

2.2 Data Structures

The data structure containing the pattern is important as it can greatly affect the complexity of change measures (e.g., can the extremal points be found in $O(n)$?). Worboys and Duckham [64] provide a useful overview of some of the common structures and their properties.

Grid-based structures are a simple starting point. The underlying concept is the *bucket*, a contiguous memory location, that will contain only points that the grid deems to be

related. The basic grid type is the fixed grid structure, in which the grid partitions the region containing the pattern into equal sized cells and each cell constitutes its own bucket. All points within a specific cell are held in the same place. The obvious problem with this is that when the dot pattern is not uniformly distributed, some cells may be empty while others may be near overflowing. An extension to the fixed grid is the grid file, in which the horizontal and vertical lines making up the cell divisions do not have to be equally spaced. They are placed based on the dot distribution and cells can be divided or amalgamated depending on the amount of free space they contain. The major benefit of the grid file is the ability for it to be easily dynamically updated, however using it to search for specific dots (extremal, median, etc) is not particularly fast. One of the more common uses of grid layouts is in the quadtree, which divides the space up into 4 evenly sized quadrants (buckets) and iterates over each quadrant performing the same space dissection. Fig. 2.1 demonstrates an example of this division and shows the tree that would be used to find any specific point. The quadtree requires both a minimum size for its final quadrants and a bounded space to delineate.

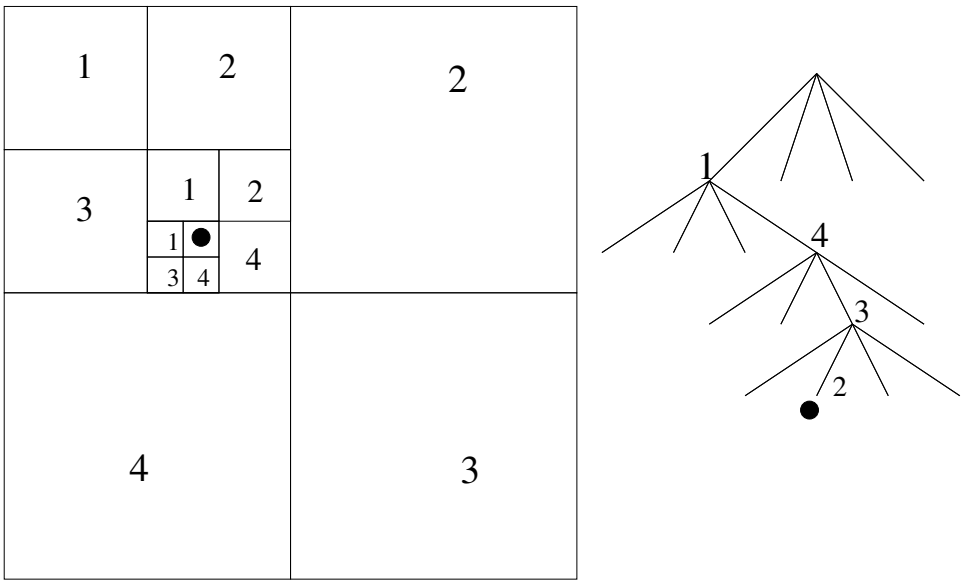
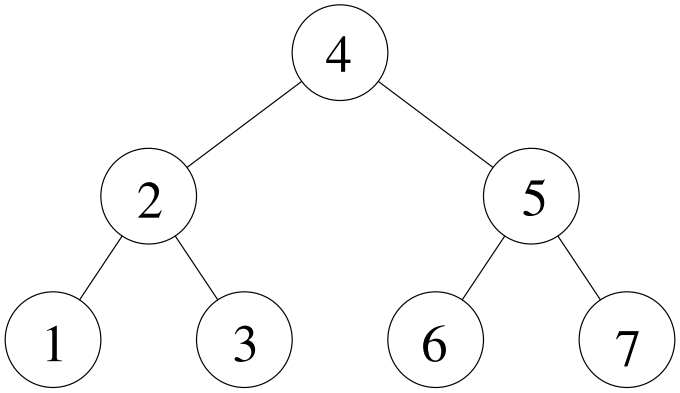


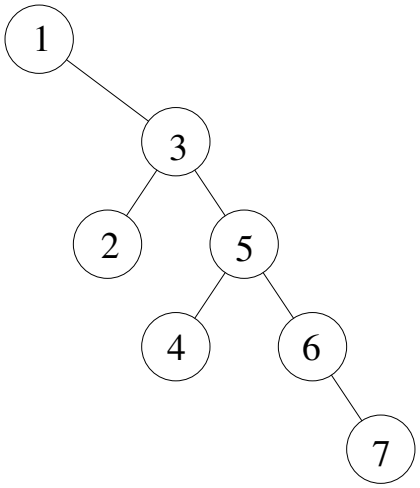
Figure 2.1 An example quadtree division that shows the route taken to find a given dot
.

Tree structures are quick to both build and search and do not have to mimic the spatial relationships of the dots, the data can be stored in any way that preserves those relationships. Knuth's book on sorting and searching [43, ch. 6.2.2-3] provides a description of the various methods that can be used to store data in tree structures, the most basic of which is the binary tree. A binary tree has nodes with at most two children and stores the data so that searching on it performs a binary search. The binary tree is fast to search but suffers from difficulties in insertion; it relies on the incoming data to be suitably randomised otherwise it builds unbalanced trees. An unbalanced tree has subtrees from the same node with unequal heights and can result in linear search complexities. Examples of a balanced and unbalanced binary tree can be seen in Fig. 2.2(a) and Fig. 2.2(b) respectively.

To prevent the creation of degenerate trees there have been a number of variations on the binary tree, further information on these can be found in Knuth [43] Worboys and



(a) Balanced Binary Tree.



(b) Unbalanced Binary Tree.

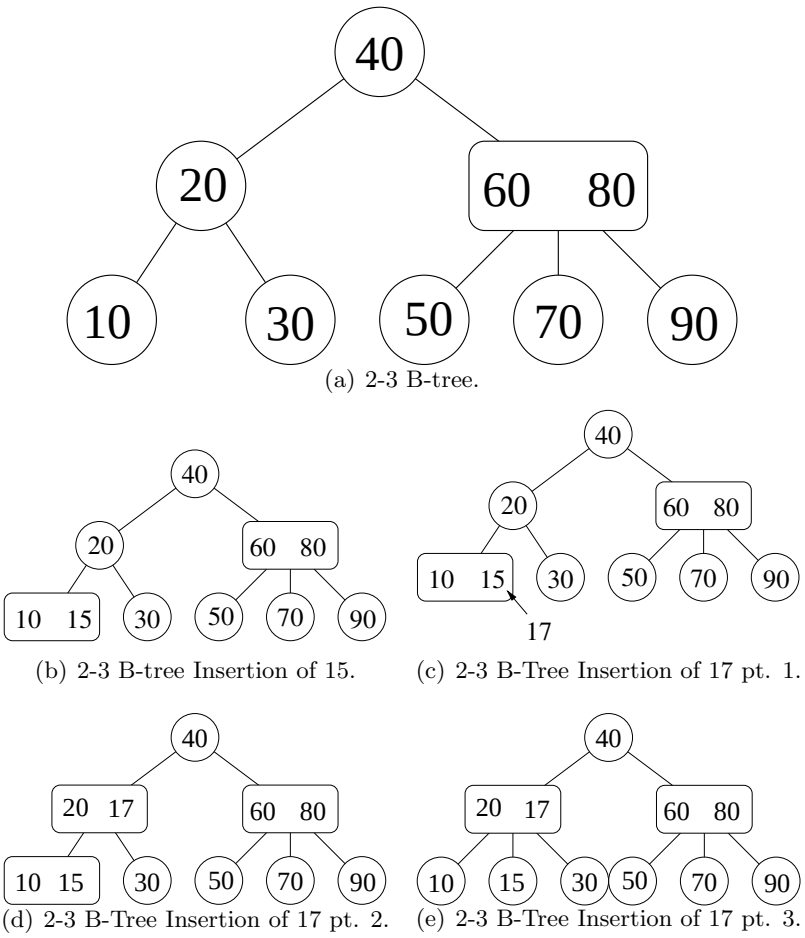
Figure 2.2 Examples of the Binary Tree Data Structure

Duckham [64] and Berg *et al.* [10]. For the purposes of this thesis we look at the 2-3 B-tree[2] in which each node of the tree can have, either one data object and two children, or two data objects and three children. This tree is far easier to balance as insertion is greatly simplified, as demonstrated in Fig. 2.3. At Fig. 2.3(b) the 10 node has space so accepts 15, however at Fig. 2.3(c) it is incapable of accepting 17 so it moves it up to its parent node, as shown in Fig. 2.3(d); as this node now has two elements it must have three children so the bottom-left node is split (Fig. 2.3(e)). This insertion approach gives a *self-balancing* tree so the degenerate branches of a normal binary tree can be avoided.

Guibas and Sedgewick [35] showed that the 2-3 B-tree can be refactored into a binary tree called a *red-black* tree[3]. The ability to mimic the 3-node of a 2-3 B-tree is achieved by having two types of linking edge: red and black, often the nodes will be referred to by the colour of edge that connects to them (i.e. red or black nodes). The black edges are the same as a standard *vertical* node linking in a 2-3 tree. However, the red edges are *horizontal* links, that is to say they indicate a link between two nodes that would be concatenated in an equivalent 2-3 structure.

[2]The source of the B in B-tree is somewhat of a mystery but is commonly attributed to either Bayer (who, along with McCreight, created the B-tree [7]) or Boeing (where Bayer and McCreight were working when they created it).

[3]They also show how B-trees of higher orders can be converted but for this thesis the description of the 2-3 is sufficient



(a) 2-3 B-tree.

(b) 2-3 B-tree Insertion of 15.

(c) 2-3 B-Tree Insertion of 17 pt. 1.

(d) 2-3 B-Tree Insertion of 17 pt. 2.

(e) 2-3 B-Tree Insertion of 17 pt. 3.

Figure 2.3 Examples of the 2-3 B-tree Data Structure

The dot patterns we consider in the thesis are generally in a 2-dimensional space and are therefore described by two values. The tree structures given above currently only sort on one value, and, while this may be sufficient for the needs of the change identifiers, it is wise to consider ways in which we might better sort the data. Berg *et al.* [10] gives a good treatment of such structures, beginning with the kd-tree. The kd-tree sorts by alternating dimensions, for example for a 2-dimensional dot pattern the 2d-tree (sometimes called a 2-dimensional kd-tree) splits alternately by the horizontal and the vertical. Fig. 2.4 shows an example in which a dot pattern has been organised into a 2d-tree, the numbers 1–4 represent splits and the letters *a*–*e* represent dots. The kd-tree is a useful structure as it

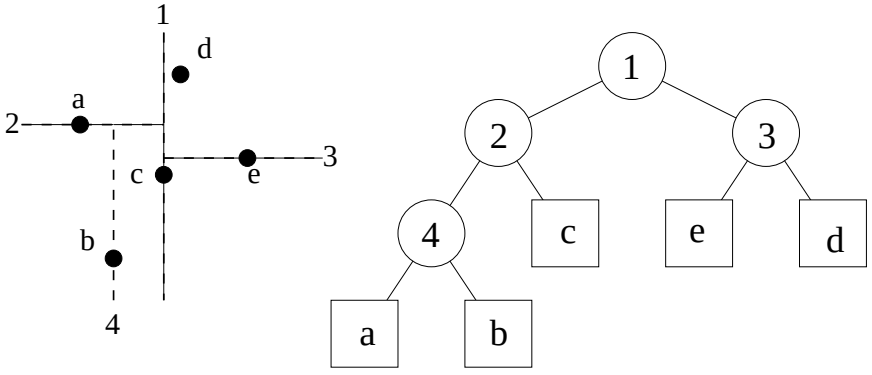


Figure 2.4 A Dot Pattern with Associated 2d-Tree.

makes searches within rectangular regions in $O(\sqrt{n} + k)$ time where n is the number of dots and k is the number of dots found in the query. Whether or not we will need to be able to perform such a query with the change identifiers will not be known till we have examined them further in Chapter 5[4].

The concept behind the general framework is that it remains applicable regardless of application. This generality requires it to be free of too many assumptions about the data, in particular no assumptions can be made about the format in which the data arrives. Some applications may simply give locations for all dots in each phase whereas others may provide information about only the dots which have moved. Not knowing, or being able to specify, the specific data format means that the data structure will likely have to be rebuilt at each timestep. Some form of balanced tree-structure would make a good preliminary choice as it would be quick to build and easy to search, however without having looked at the possible change identifiers' requirements it is impossible to know which specific structure would be best. Consequently, this will be revisited in the chapter detailing the methodology used for the experimentaion (Chapter 6).

2.3 Spatio-Temporal Data

The existing approaches to spatio-temporal data include both qualitative and quantitative methods. The change identifiers are primarily a quantitative concept but by looking at the qualititative as well as the quantitative we provide a better background for our own research. It should be noted that this is by no means an exhaustive literature review as the field of spatio-temporal research is both extensive and very much alive.

2.3.1 Qualitative Representation of Change, Hornsby and Egenhofer

Hornsby and Egenhofer's paper [37] begins with a definition of objects and a discussion on object identities; drawing attention to the fact that, when the members of a collective can move and change, identity is not always easy to ascertain. This identity ambiguity is of particular importantance to their work as the changes that they consider are directly related to an object's identity. The work provides a visual representation for describing different change types as a product of identity operations on objects and an extension of this to describe the change in object composites. Identity and its relation to dot patterns will be addressed in Chapter 3; for now it suffices to state that a dot patterns identity is inextricable from its component dots and the timestep for which it was created. The work in this thesis is primarily concerned with quantitative descriptions of change in which distance is important, as opposed to the topological approach. One possible avenue for further research may be to look at using change identifiers to indicate topological changes.

[4] Berg *et al.* also describe the range trees which are better suited to rectangular range queries when there are many dots that will be found (i.e. when k is large)

2.3.2 A Spatio-Temporal Taxonomy for the Representation of Spatial Set Behaviours, Thériault *et al.*

Thériault *et al.* [58] present a taxonomy for describing evolutions of sets of geographical entities (SGEs). The authors make clear that there are two simplifications on which their model rests: Firstly that a geographical entity exists in geographical space; and secondly that the size and orientation of the entities is negligible compared to the inter-entity relationships. These simplifications restrict the behaviours that they examine to changes in a point-based framework. The point-nature of an SGE means that, for the purposes of this thesis, we can treat it as largely identical to a dot pattern. Thériault *et al.* are primarily concerned with the manner in which SGEs can change, and to do so they need properties on which the change can be measured. It is these properties that we are interested in examining as classifiers for describing patterns.

Before discussing the measurements that they propose we shall draw attention to an interesting observation that the authors make. There are two different but complementary method types for examining the evolution of entities.

1. Deductive methods which are based on representations of the spatial entities and their relationships. They note that these methods tend to be based on Euclidean space and/or topological descriptions of space and time. Thus deductive methods form a qualitative approach.

2. Inductive methods; methods based on data analysis often using spatial statistical methods to study properties and distributions of the entities. In contrast to deductive, inductive methods are a quantitative approach.

While laying the groundwork for the taxonomy the authors reason about descriptions of possible complex set behaviour arising from simple entity behaviours, however, like the work presented in this thesis, the majority of Thériault *et al.* 's paper takes a low-level approach; one that is concerned with the measurable changes that take place in the set.

Further to the point-nature they use for the entities of the sets they assign an intensity to each entity indicating the importance the point has within the given context. Such properties for the inquiry presented by this thesis for the fact that we allow for dot patterns that arise from contexts in which intensity is not a meaningful concept, for example, a graph of classification data where each data-point is a dot in our pattern.

The taxonomy itself arises from the consideration that the entities can only exhibit very simple changes: appearance, disappearance, translation and intensity change. These changes are manifested within the set as four components of possible change types (not a necessarily exhaustive list). It is these components which bear such close resemblance to our descriptors.

1. **Territorial/Spatial Extent**
   Thériault *et al.* use the phrase territorial extent; however, because of the above-mentioned concerns about applications outside of GISc it makes more sense for this

thesis to refer to spatial extent. The extent of the SGE can change via expansion or contraction. To measure the extent the authors suggest using the convex hull of the set. This is assigning a footprint to the set and taking a measurement from that footprint. This topic is re-visited when dot pattern descriptors are outlined, as using a surrogate for the dot pattern has some interesting connotations for the types of measurement that can be made. Extent is clearly a property that will need to be accounted for within the set of descriptors.

2. **Spatial Distribution**
   This component of Thériault *et al.*'s change types includes several sub-properties. Of the four properties of spatial distribution all but the last have direct correlation with some of the descriptors we introduce in Chapter 3.

   a) Centre of Gravity (CG): Indicating the equilibrium point of the distribution. The CG takes into account the intensity of their dots but aside from this difference such a measure is used within the framework presented by this thesis as a descriptor of position.

   b) The Standard Distance (SD): The standard deviation from the CG. Like centre of gravity this quantitative measure has a descriptor counterpart; standard deviation can be used as a measure of the extent of a dot pattern.

   c) The Orientation: Using principal axis extraction the direction in which the set can be said to face can be found. Orientation is one of the classes of descriptors used by this thesis and the principal component's (axis') gradient is one of the descriptors within the orientation class. Thériault *et al.* use both of the returned vectors of the principal axis detection[5] so that they can describe the maximum and minimum dispersion of the set. This is then used by the fourth and final property of spatial distribution.

   d) Ellipse of Dispersion: Defined by the two vectors given by the principal axis extraction, this ellipse is used to monitor overall dispersion and density. Thériault *et al.* note that this is an extent measure that, unlike the convex hull, is not sensitive to outliers within the set.

3. **Spatial Pattern**
   Spatial pattern refers to distinguishing between random, clustered and regular patterns. How best to measure this is a topic that we will consider in Chapter 3, for now it can be thought of in terms of degrees of homogenity across the pattern.

4. **Spatial Autocorrelation**
   The final component concerns the relationships between the entities of the set. It is used to measure how likely entities are to have the same or similar attributes to nearby neighbours. As the patterns we consider have no attributes beyond their location this can be dealt with by the same kind of approach as for clustering analysis.

[5] Principal axis extraction, or Principal Component Analysis (PCA), in 2-dimensions returns two perpendicular eigenvectors, the one with the highest eigenvalue is the principal axis

The taxonomy presented by Thériault *et al.* shows a strong resemblance to the kind of analysis we wish to perform on dot patterns and certainly has a bearing on the descriptors we have chosen. However it is concerned with changes that can occur to the set and has not been created to describe the pattern as it is at any time step. Part of our research is to see if other ways of identifying change arise from examining the properties inherent in a static pattern. We have examined Thériault *et al.*'s paper in more detail than we have done for many of the other works as it is so close in nature to the aspect that concerns our research.

2.3.3 Granularity in Change Over Time, Stell

Stell [57] examines the levels of detail at which entities undergoing change can be modeled. Within this thesis it is assumed that the granularity of the phases of the dynamic dot pattern is appropriate to the application context, and hence is not explored further. However when considering granularity Stell defines the nature of the *time domain* over which the data exists, and this is directly applicable to our work. A time domain is a finite set T such that for $t, t', t'' \in T$:

$$t \prec t' =_{def} t < t' \wedge \neg\exists t''(t < t'' < t')$$

In effect $x \prec y$ asserts that x and y are adjacent and that x precedes y. We can use this same requirement for the timesteps within the time domains used by our dynamic dot patterns. As each timestep has an associated phase of the dynamic dot pattern it can be stated that a phase ϕ' precedes a phase ϕ'' ($\phi' \prec \phi''$) iff the relationship is also true for their associated timesteps.

$$\phi' \prec \phi'' \iff t' \prec t''$$

Note that we allow for a 'live' system, in which T may be an infinitely large set[6]. To clarify what is meant by a 'live' system envision an application monitoring a herd of cows in real time. The change identifier framework sits in an idle state awaiting herd data; processing the dot patterns as they arrive.

Given a time domain T, Stell defines a *dynamic set* over T as a set of objects that evolve over time. He uses dynamic sets to explore the concept of support amongst entities; if entity a supports b then a exists/ed prior to b and the existence of a is necessary for the existence of b. The support relation and the time domain allow Stell to produce graphs to model the relations between entities over time, and simplify these graphs by the almagamation of entities or by omission of time steps.

The overview given of Stell's paper is by no means an exhaustive examination of its content. It suffices for this thesis to note the added formalisation it provides for the time domains that support the dynamic dot patterns.

[6] Although, for obvious reasons, all our tests have a finite size and it is possible to make the argument that no real world example is infinite. Despite this, when using the framework it is not known a priori when, if ever, the data will cease to arrive, so assuming infinity leads to a more general framework.

2.3.4 Finding REMO - detecting relative motion patterns in geospatial lifelines, Laube *et al.*

Laube *et al.* [44] is one of the core works concerning motion analysis. Laube *et al.* define the concept of *geospatial lifelines* as a series of observations on Moving Point Objects (MPOs) that are a triple of ⟨id, location, time⟩; an individual dot within a specific phase of a dynamic dot pattern can be described by the same tuple. The main research interest of this paper is identifying flocking behaviours (and other motion patterns[7]) of MPOs by analysis of spatially constrained RElative MOtions (REMOs). These movement patterns are the same complex behaviours that we have discussed previously as the behaviours of the collective which the dynamic dot pattern represents. One of the issues faced by REMO analysis is the complexity of some of the pattern identification algorithms. It is possible that the change identifiers can provide information on when a movement pattern type has occurred or changed thereby reducing the number of times which the analysis needs to be run.

2.3.5 Event-oriented approaches to geographic phenomena, Worboys

Worboys [63] gives an alternative view of the concepts behind spatio-temporal data analysis. Worboys states that instead of looking at entities at each time step the phenomena should be viewed as sets of events. The events in this context are 'happenings' unlike the events as objects described by Sullivan and Unwin [51]. While this approach is outside the remit of this thesis it is worth noting that it is only convention which sees us mapping the entities of a collective in our dot patterns.

Worboys, when discussing existing ways of looking at object change, gives a description of the problems with dot pattern identity. Like Hornsby and Egenhofer [37], Worboys notes that identity is not necessarily a fixed concept. Both Worboys and Hornsby & Egenhofer observe that there is an issue when assigning identity to a dot pattern when the membership of the pattern is subject to change[8]. This uncertainty is one of the reasons that the dot patterns, as proposed by this thesis, are independent of a personal identity and why the change identifiers function whether or not identity information is known for the dots.

2.3.6 Reporting Flock Patterns, Benkert *et al.*

Benkert *et al.* [8] provide a set of algorithms for identifying movement patterns of the same type defined by Laube *et al.* [44]. The algorithms proposed make use of the skip-quadtree, a data structure that is an extension of a standard region-quadtree[9] but that only stores boxes/buckets as leaf nodes if they contain at least one point. This reduces the space

[7] Note that a motion pattern is different to a dot pattern. To avoid confusion we shall endeavour to avoid using pattern without an appropriate classifier.
[8] An analogy of the identity issue is provided by the philosophical problem presented in the Ship of Theseus
[9] Called just a quadtree when it was described earlier.

required to $O(dn)$ (Where d is the dimensionality of the space the data exists within) at a cost of increasing the time to check if a bucket is empty.

In particular Benkert *et al.* focus on identifying when a group of entities constitutes a flock. They provide two flock definitions, both of which rely on being able to find an encompassing disc of the entities; this disc is a footprint. The definitions differ in the granularity of time intervals at which they require a disc to contain all points; the first requires a disc to be found at all time points whereas the second only requires such a disc at the discrete timesteps[10]. They present two algorithms using the both the skip-quadtree and these definitions to detect flocks. The working of these algorithms is of particular interest to the work within this thesis as they allow for an approximation in their disc radius. There is a conceptual similarity in observing that the footprint need not be exact. It should be noted that theirs is always an over-estimation; Benkert *et al.*'s approximation is performed by adding a positive uncertainty to the radius of the disc.

2.3.7 Designing visual analytics methods for massive collections of movement data, Andrienko and Andrienko

Andrienko and Andrienko's 2007 paper [4] presents a set of tools for the visual analysis of large amounts of movement data (so large that it exceeds available computational memory). Within discussion of this topic they raise certain concerns which are pertinent to the research presented in this thesis. To begin with they define movement data as a function that matches a tuple of ⟨entity, time moment⟩ (like Laube *et al.* [44]) to a point in space. Bearing a resemblance to the approach within this thesis is Andrienko and Andrienko's decision to examine the information that can be gathered from a simplified starting point. They, however, forgo looking at the static pattern and focus on what they call the *derivative movement characteristics* (speed, duration, etc.). These derivative movement characteristics combined over time lead to *individual movement behaviours* (IMBs) which are the more complex movement types indicative of the underlying collective's entity behaviours. These definitions lead to the concept of *momentary collective behaviours* (MCBs) which are the movement characteristics of the set of entities at a time moment. MCBs are close in nature to our descriptors, looking at the spatial and statistical distributions of the entities and their characteristics. Over time MCBs give rise to the *dynamic collective behaviour* (DCB); the overall description of the set's complex behaviour.

Andrienko and Andrienko focus more on the classification of these behaviours and how to identify them than on an in-depth examination of the change the sets may undergo. The reason we give such a detailed description of their definitions is the influences they note that can affect the behaviours. These influences are not something that affect the construction of the change identifiers but are of absolute importance to any application that may use them:

[10] They also demonstrate that, if the entities within a set move along straight line segments between consecutive positions, then the flocks produced by both definitions are equivalent

- Properties of space (e.g., altitude, accesibility, function, etc.)
- Properties of time (e.g., temporal cycles, duration of daylight, holidays, etc.)
- Properties of entities (e.g., age, movement method, purpose, etc.)
- Various affecting phenomena (e.g., climate, sport, culture, etc.)

Should an application's users be fully aware about all of these factors it is likely that they will be able to choose change identifiers that check for change in ways that are appropriate to the context.

Further to their discussion on factors that can affect the data, Andrienko and Andrienko note that correlation of two types can affect the DCBs; that by *influence* and that by *structure*. Influence correlation is when one characteristic directly affects another whereas structure correlation occurs when two or more characteristics are combined to form a new more complex characteristic. We examine how correlation affects the dot pattern descriptors (and therefore the change identifiers) in Chapter 3.

While visualisation is not the core of this thesis the change identifiers could add to Andrienko and Andrienko's work by providing a concrete examination of how the patterns can be assessed.

2.3.8 Towards a Taxonomy of Movement Patterns, Dodge *et al.*

Dodge *et al.* [19] provide a formalised approach to describing the movement of patterns. The authors define the patterns for which they supply a taxonomy as consisting of *Moving Point Objects* (MPOs), which are dimensionless entities, similar to our dots but with movement data associated. They also provide a formalisation of movement that uses three groups of *primitive parameters* and their derivatives. Dodge *et al.*'s full classification is extensive and there is not the space to describe it here in detail, instead we will discuss the fashion in which they have sectioned its levels. The movement patterns[11] are split into *generic* and *behavioural* patterns, generic being the most widely applicable term for a movement type, for example periodicity is exhibited by any MPOs that have periods of movement punctuated by periods of largely static behaviour. Behavioural patterns, however, are far more specific movement types such as migration or fighting. The generic patterns are split into compound and primitive types, primitive for patterns where only a single movement parameter changes and compound when more than one change occurs. All the given pattern types can apply to individuals or groups. As with previous work focusing on the complex movement behaviours, Dodge *et al.*'s work does not directly affect the change identifiers. It is important, however, to have a good understanding of the behaviours that underly change so that it can be assured that no particular form of change is missed.

[11] Patterns here meaning behavioural patterns and not spatial distribution patterns

2.3.9 Modeling Herds and Their Evolvements from Trajectory Data, Huang *et al.*

Huang *et al.* [38] present a set of four evolutions that a herd can undergo: expansion, joining, shrinking and leaving. These evolutions do little to describe the pattern at any single time, however, instead describing the changes in state that the herd has undergone. This is used, primarily, to provide a way to define the identity of a changing herd. One comment they make that is of particular relevance is that quantitative measures can become qualitative if a significant change has occurred. This leads to the problems inherent in trying to describe where the boundaries of significant change are, but may indicate a way in which change identifiers can be best used to provide qualitative information.

2.3.10 A taxonomy of collective phenomena, Wood and Galton and Detecting and Identifying Collective Phenomena within Movement Data, Wood

Wood and Galton [62] build on previous taxonomies ([4, 19]) so that the concepts behind a collective are tightly defined. The definition of a collective they provide involves six observations, given these observations they then have a concrete base from which to form the criteria for their classification. Wood [61] extends and uses this classification to identify collectives from within spatio-temporal data. Dynamic dot patterns can be seen as an abstraction of a collective so, while the criteria do not directly apply to dot patterns, it seems prudent to examine them so that we can be confident that no important information is left out by our abstraction. Wood and Galton provide a set of considerations that are used by their classification, but going over each in detail would be irrelevant for this thesis. Instead only those spatial aspects which relate to dot patterns will be covered. Before looking at these considerations we note that, like Worboys [63] and Hornsby and Egenhofer [37], Wood and Galton discuss whether a collective maintains identity if the cardinality or identity of its members changes.

Location

By convention the location of a dot pattern tends to be taken as a point, often the centroid (mean position) of the pattern (the Centre of Gravity used by Thériault *et al.* [58]). The collective definition presented by Wood and Galton is different in that it treats the location as an area or volume. For example the location of a class can be said to be the classroom and, from the level of granularity of the class itself, this constitutes a real 3-dimensional space. They suggest that one way of finding this location could be by aggregating the footprints of the collective at each time step of its existence.

The location classifier is interested in distinguishing between collectives by the fashion in which they change their location. This thesis is more concerned with being able to classify between different types of dot pattern without necessarily knowing the methods by which

they might change.

Coherence

Coherence as it appears in [62] relates to the exhibited behaviour(s) of a collective and can arise from two main sources: *cause* and *purpose*. Causes are split into external and internal sets; the example given by Wood and Galton of an external cause is of Earth's gravity causing raindrops to fall as a collective, and the example they provide of an internal cause is the mutual gravitational pull of a star cluster maintaining the collective of stars. Purposive collectives maintain their collective nature via some goal, again this can be an internal goal assigned by members of the collective or a purpose placed on the collective by some external agent. Wood extends the discussion on coherence in [61] by defining some coherence criteria, of which a group of individuals must satisfy at least one to be considered a collective. It is these coherence criteria which allow Wood to be able to identify spatial collectives from spatio-temporal data.

The change identifiers as used within this thesis are not concerned with the coherence of the dynamic dot pattern. However further work could examine the elements of coherence exhibited by dynamic dot patterns to see if new change identifiers are suggested.

As was described in the introduction, neither a dot pattern nor a dynamic dot pattern is a collective; the collective may be a pod of whales but the dot pattern representing this is a snapshot of whale positions at an individual moment and the dynamic dot pattern is a sequence of the dot patterns. The classification allows us to provide further distinction between a collective and its representative pattern(s): The collective classification allows for change within its description at different time steps; it can cover a broad time period with complex definitions covering various phases in a collective's life span. For a pattern we are interested in describing it as it is (or was) at the time the information was recorded and how it may differ in type from another pattern. This allows avoid the question of identity for a dynamic dot pattern to be avoided; it can be assumed that phases of a dynamic dot pattern are from the same collective in that they are all in the collective for which the dynamic dot pattern was created to represent. Tracking a constant collective identity over real world entities is more complicated as issues of changing membership arise (also discussed by [37, 63]).

As with the taxonomies provided by Andrienko and Andrienko [4] and Dodge *et al.* [19], Wood and Galton's classification describes the high-level behaviours that may be exhibited by the entities that underly a dynamic dot pattern. It is possible that our change identifiers will indicate these behaviours and the changes between them. Even if this is not possible we would be remiss not to try and understand the possible reasons for the change measured by the identifiers and such analysis may also inform the creation of new identifier types.

2.3.11 A Graph Model for Spatio-temporal Evolution, Del Mondo *et al.*

Del Mondo *et al.* [17] discuss the shortcomings in using only space-time paths to describe the events, processes and changes an entity can undergo. They propose using a graph model to map the complex networks that can be formed. To this end they note three relation types that must be modelled:

Spatial – Relation between two entities at the same time

Spatio-temporal – Relation between spaces occupied by entities at differing times.

Filiation – How entities at distinct times relate to each other (i.e. descent or transmission).

Such a model provides a view of a group of entities in a space that does not lose information about their relationships and allows route tracking of entities through different states.

The work is interesting as an examination of visualising change and the collectives underlying dynamic dot patterns can certainly be drawn in such network graphs. The change identifiers may provide information about salient timesteps at which the network graph could (or should) be updated so that for large, and long, dynamic dot patterns the network graph would remain a manageable size.

2.3.12 How fast is a cow? Cross-Scale Analysis of Movement Data, Laube and Purves

Laube and Purves [46] is an unusual paper. Whereas the other works by Laube mentioned here are examples of movement analysis, this paper looks at the concepts and problems inherent in analysing movement data, akin to the influences in Andrienko and Andrienko's paper [4]. Most of the concerns they raise are not directly related to the general framework we propose as they are application-dependent. However, some are relevant to the test data used for this thesis and some will apply to any application that uses the change identifiers. While there is not the space to discuss each point in detail, a general overview will be given of their concerns (leaving the amusingly alliterative appelations applied by Laube and Purves intact).

Granularity Grief: Any measured parameter (like change identifiers) will be greatly affected by the granularity of the time domain. The sampling rate at which the data is provided can result in lost or misleading information. This point is similar to Andrienko and Andrienko's *properties of time* influence. It also bears a relation to Stell's consideration of granularity modelling [57].

Slippery Spaces: The space in which the entities exist may not be unconstrained Euclidean space, for example a city has paths, roads, buildings and pre-determined crossing points. This point is similar to Andrienko and Andrienko's *properties of space* influence.

Delusive Dwarfs: Scaling and sampling may provide faulty data. While computationally easier to process, small data sets do not necessarily indicate the behaviours of larger sets. This is something that must consider when drawing any conclusions from the real-world test data and the generated test patterns used in this thesis.

Baffling Bias: The data source may be biased. For example some types of people are more likely to allow themselves to be tracked than others. Such people may have movement patterns dissimilar to others. Bias bears a similarity to Andrienko and Andrienko's *properties of entities* influence and their *affecting phenomena*; specifically their observation about differing cultures.

Cast-off Context: Often work in the spatio-temporal and GISc fields looks only at the entities in relation to each other and ignores the geographical context in which they are positioned.

Sinful Simulations: When making test data sets it can be difficult to find the appropriate balance between randomness and realistic movement. This is obviously a concern for us within this thesis and we look at the source of the dynamic dot patterns used for testing in the chapters on dot patterns and change identifiers (Chapters 3 and 5 respectively).

2.3.13 Others

There was much work that focuses on spatio-temporal data that was read as background for this thesis has not been detailed above, as it did not add greatly to the discussion of change identifiers. However, any work read will certainly have affected our approach, so should reader wish to pursue further reading they could start with: [25, 13, 9, 18, 45].

2.4 Shape

To find descriptors for a pattern we must accept that the pattern is an entity, possessing properties emerging from its component dots but not inherent within any individual dot. Intuitively this can be done by considering a region as a surrogate for the pattern, and this assumption brings with it methods for measuring the descriptors (e.g., area as a measure of extent). We consider any region that describes a dot pattern as a footprint of the pattern. The fact that assigning a region or footprint is so intuitive is almost certainly due to Gestalt perception but it does lead to the question: Is any region more 'correct' than another if they are equally intuitive? Before looking at the footprints, some of the ways in which shape itself has been considered are explored. Not only will this provide a good background on how footprint analysis might be approached when change identifier assessment is discussed (Chapters 5 and 6) but some of the methods may apply to dot patterns without the need for a region assignment.

Galton [26, ch. 4.7.3] provides a comprehensive overview of the possible attributes and

relations of spatial regions; specifically: dimension, connectivity, position, location, orientation, size and shape (concerning spatial attributes like sinuosity).

Dimension

Galton splits dimension into two types: strict and apparent. Strict dimension is the classical approach in which an object classified as n-dimensional has no extension into any dimension greater than n. Apparent dimension is a product of having differing levels of granularity with which examine the object. A road is generally considered to be a 1-dimensional line on most maps, a 2-dimensional surface for most users and a 3-dimensional object for a road builder who has to be concerned with the depth as well as breadth and length.

Dimensionality of a dot pattern works best when considered as apparent because it is always possible to draw a 1-dimensional curve through the dots. In 2-dimensions, for example, it may be of interest how collinear the arrangement of the dots is.

Connectivity

Connectivity is not a directly applicable term to dot patterns as the dots are, by their nature, disconnected. However it is certainly true that sometimes the dot patterns have areas that appear distinct from others. Fig. 2.5 shows a dot pattern in which part a is separated from part b by a distance that is substantial compared to the inter-dot differences within each part (each part is called a *component* of the dot pattern). Such dot patterns could be said to be disconnected and identifying such separations is intuitively important.

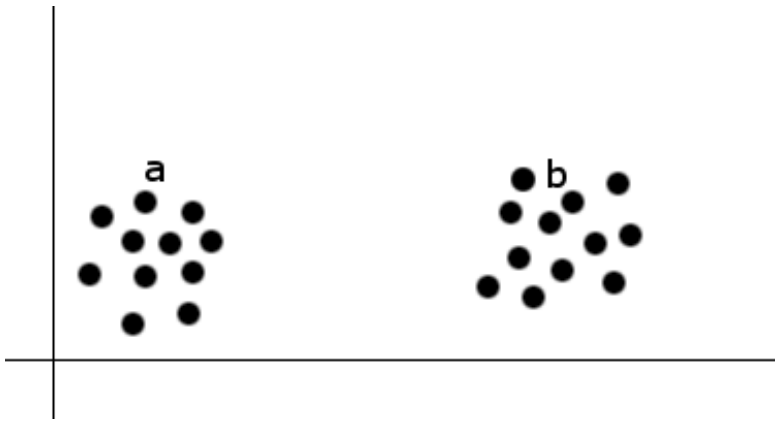


Figure 2.5 A single dot pattern showing disconnected components.

Position

Position is perhaps the most immediately obvious facet of information that can be gained from a dot pattern, as a pattern is represented within spatial dimensions. Galton notes that the problem with position is choosing the region of space from in which to base

the location reference system. As such position is split into two parts: location and orientation. For the purposes of this paper however this is not a relevant concern and Galton's definition of location will not be described in detail; suffice to say it discusses using different reference systems from which to find a 'target' object.

Orientation

Orientation is the direction in which the object can be said to face, or point. Galton describes using a directed line as a 'reference axis' such that the orientation can be found by the direction the axis points. With an axis from which to find the direction the difficulty becomes one of ascertaining how best to specify the direction; for which the book provides several approaches, both quantitative and qualitative. The quantitative methods are variations on units for measuring the line angle and are, as a result, straightforward. The qualitative methods are somewhat more interesting as they can be envisioned in two ways. Firstly by describing the direction in which the axis points to (e.g., north, south, east, west, up, down, left, right, back, forward). And secondly by describing the direction as an observer looking at the object by which sections of the object are visible (the different faces of a cube is the example given in the book). As previously mentioned, this thesis does not focus on the qualitative description of dot patterns, but Galton's quantitative methods will be looked at in greater detail when we discuss descriptor measurement methods (Chapter 3).

Size

Size is a cognitively obvious description of shape. However the fashion in which it is measured is a topic worthy of discussion. Area and volume, for example, are commonly used and often sensible measures but, as Galton points out, they reduce a 2-dimensional and 3-dimensional quantity respectively to a single value. This single value means that information is immediately lost as two different objects of differing dimensions (height/width/breadth) can produce the same result.

The book also describes angular extent as a measure of size but as this requires an observer it does not apply to the context in which we are examining dot patterns.

Shape

Galton uses this section to discuss various ways in which shape can be described including discussions on convex and concave, curvature, symmetry, and the use of natural language. None of these are applicable to the description of a dot pattern but may have use in describing footprints; one of the suggested areas for further work is to produce a more extensive taxonomy for footprint classification so that they may be compared more accurately.

2.4.1 Others

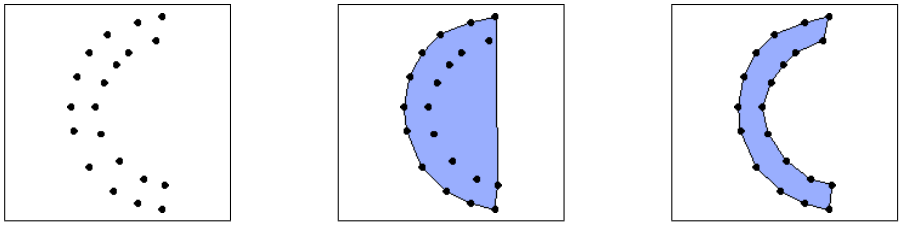
Galton's book is not the only text that details the properties of shape and more quantitative treatements can be found in the field of computer vision such as the work by Žunić and Rosin (e.g., [54, 59, 60]). However much of this quantitative work requires knowledge of the shapes' boundaries and angles, which is information that requires the footprint to have been computed. Galton's work suffices to give a general overview of the properties of shape that we can use to examine the properties of a dot pattern.

2.5 Footprints

There is a fairly large body of work about the generation of footprints, publications from as early as 1973 ([39]) presenting a variety of different algorithms to create representational shapes from dot patterns. Amongst this work there are surprisingly few that examine the footprints created in a comparative fashion. Also conspicuous by its absence is a systematic approach to determining the quality of the produced footprint. Galton [27] makes significant inroads in to both determining how 'good' a footprint is and why this is difficult to judge.

A discussion of the footprint algorithms should probably begin with one of the first to give an efficient algorithm for its computation in 1973. Jarvis [39] presented an algorithm, since called the 'Jarvis March', to generate the convex hull of a dot pattern. The convex hull is almost a base level of footprint, its algorithms are generally easily computable and it has distinct mathematical properties. Importantly the convex hull is unique for any particular dot pattern.

The convex hull is not without its problems as a representation.



(a) Point set (b) Convex Hull (c) Non-Convex Footprint

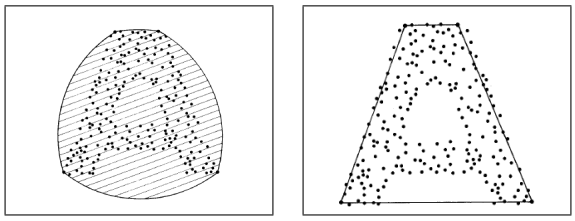
Figure 2.6 When a convex hull is inappropriate

The point set given in Fig. 2.6(a) could reasonably be interpreted as forming a 'C' shape. The cavity that dictates this shape may be important for the application context (e.g., reconstructing text from samples of a document image) and is lost when the footprint is the convex hull Fig. 2.6(b). For the given application Fig. 2.6(c) is a better approximation of the underlying data. An algorithm capable of reaching a better fit representation is a non-trivial problem and one of the earliest, and much-referenced, papers on the subject is by Edelsbrunner *et al.* [23]. 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. Taking a half-plane to be a closed disc of infinite radius, an α-hull can be defined as the intersection of all closed discs with radius $1/\alpha$ that contain all the points of S. Using a radius of $1/\alpha$ allows the convex hull to be produced when the arcs are sufficiently straight. If we assume that if $\alpha = 0$ then $1/\alpha = \infty$ [12] and that an arc with an infinite length radius is a line, we can guarantee production of the convex hull when $\alpha = 0$. These assumptions are integrated into the description of the α-hull using the idea of a *generalized disc*[13]. A *generalized disc of radius* $1/\alpha$ is defined as a disc of radius $1/\alpha$ if $\alpha > 0$, a halfplane if $\alpha = 0$ and the complement of a disc of radius $-1/\alpha$ if $\alpha < 0$, the α-hull, then, is the intersection of all closed generalized discs of radius $1/\alpha$ that contain all the points of S.

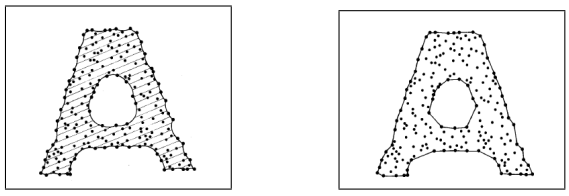
Before the α-shape can be defined some properties of the hulls need to be noted. A point p from the set S is an *α-extreme* if there exists a closed generalized disc of radius $1/\alpha$ such that p lies on its boundary and it contains all the points of S. If two α-extreme points can share the same generalized disc then they are said to be *α-neighbours*. The α-hull is the intersection of these discs (Fig. 2.7(a)). The α-shape is the straight line graph with vertices at α-extreme points and edges connecting the α-neighbours (Fig. 2.7(b)).

The positive α-shape (where $\alpha > 0$) is clearly a footprint, notable in that it tends to look like an approximation of convex hull save that it is possible for it to not contain all the points Fig. 2.7(b). However the negative α-shape (where $\alpha < 0$) produces far more interesting results as shown in Fig. 2.8(b).



(a) α-hull (b) α-shape

Figure 2.7 Images show the difference between α-hull and α-shape. Image from [23]



(a) α-hull (b) α-shape

Figure 2.8 Negative α-hull and α-shape. Image from [23]

This is one of the earliest steps toward an algorithm that is cognitively more 'appropriate' for the dot pattern than the convex hull. There is one more facet introduced by this paper that is of general interest when considering footprints. They make note that as α changes there are only a finite number different α-shapes that can appear[14]. These finite

[12] $\alpha \to 0, 1/\alpha \to \infty$

[13] American generalized instead of the British generalised as this is how it is given in the paper.

[14] Although there can be infinite α-hulls with differing curvature, the α-shapes have straight lines joining the vertices

shapes are bounded by the convex hull at one extreme and the 'null' footprint in which no dots are connected. This range of shapes is called the *shape spectrum* ($SP(S)$), and by generalising their algorithm the $SP(S)$ can be found in $O(n \log n)$ time. The examination of the literature in this section will concentrate on this type of analysis, however, aside from the papers presented here, it should be noted that such discussion on the nature of the footprints is uncommon. Edelsbrunner *et al.* do not comment on how the inherent properties of the dot pattern affect the shape produced, nor do they present any discussion on how to choose α to produce a specific shape from the shape spectrum.

Since this landmark paper much use has been made of α-shapes, in [24] Edelsbrunner and Mücke make note of two of the most interesting applications, namely molecular structure mapping and distributions of a point set. The paper presents an extension of α-shapes into 3D, but they have also been extended to take into account any intrinsic weighting of the point set in [22].

α-shapes require a parameter from which to be formed, as discussed in the introduction this is common amongst all the *non-convex* footprint algorithms. This parameter is required because the idea of the footprint is vaguely defined; any shape that can be said to represent the underlying dot pattern is a valid footprint. The reason that footprint as a concept remains vague is that users of the algorithms have different requirements on the type of shape they need. The parameterization allows control over the detail captured/lost within the footprint therefore allowing for different footprints to be created for different applications.

There are many possible algorithms besides the α-shape that we could examine, however rather than listing them (for a larger study of existing algorithms see [21]) the rest of this section will look for novelties within the literature.

Melkemi [47] suggested an alternate approach to a single value parameter, such as α used by the α-shape, for their $\mathscr{A}$-shape[15] algorithm: the parameter ($\mathscr{A}$) is a set of dots. The footprint is then constructed from the Voronoi Diagram of the union of the original dot pattern and $\mathscr{A}$; the footprint is defined by the outer borders of the cells containing the original dots. The process for choosing $\mathscr{A}$ is not expanded on until [48] in which it is defined as sampled from the union of two sets:

1. The centres of the Delaunay circles associated with the Delaunay triangulation of the original pattern having radii higher than a threshold $\tau \geq 0$.

2. For each edge $\overrightarrow{pq}$ of the convex hull of the original pattern, consider the point not belonging to the convex hull of S and which is the centre of the circle passing through p and q and having sufficiently big radii.

What is meant by 'sufficiently big' in the second constraint is not elaborated on, given the nature of the work we can assume it relates to the same threshold as in the first constraint. Melkemi and Djebali [49] introduce the idea of the weighted $\mathscr{A}$-shape, this

[15] In [47] it is referred to as the $\mathcal{A}$-shape but in [48] and [49] it is called $\mathscr{A}$-shape, so we use the most common notation

allows the algorithm to deal with dot patterns containing areas of different densities. Each point is given a weight based on the distance between it and its closest neighbour. The set of points $\mathscr{A}$ is found using what they call 'the power diagram' of the original pattern, suffice to say that it too uses a threshold value much the same as the unweighted version.

Alani *et al.* [2] also use a point set as their input parameter but their paper differs from others in the field in that it is one of the few where the application has directly led to the development of the algorithm. There exist gazetteers (or geographical thesauri) which combine place name data with limited locational information. Such systems are used for queries such as requests for all the hotels in a specific area. After noting some of the current constraints on such systems (limited bandwith, differing search terms to index terms, imprecise or precise matching, etc.) they introduce the *Dynamic Spatial Approximation Method* or DSAM. Much the same as the work performed by Melkemi and Djebali, it uses the union of the original dot pattern and another set of dots to construct the Voronoi Diagram, the footprint being the union of cells containing the original pattern. Unlike the $\mathscr{A}$-shape, the external points are already in the database as points known not to exist within the query location. In this instance, as the data has already been obtained, there is no need to add the further complexity of sampling. The aim of DSAM is to approximate a known area, the previous algorithms covered are not so specific, and this allows Alani *et al.* to 'score' the footprints that DSAM generated. They give three methods with which to evaluate their approximation:

- Total areal error – Gives a basic approximation error.
- Visual error – Gives a measure of how different the shapes are. If we take the negative false error as the areas left out of the approximation and the positive false error to be the areas in the approximation not in the expected area then the visual error can be measured as:

  $$V = \frac{A_{pp} + A_{np}}{A_o} 100\%$$

  Where V is the visual error, A_{pp} is the positive false error, A_{np} is the negative false error and A_o is the original (known) area. A similar technique is used in this thesis; the quality of the change identifiers is assessed by the difference between the footprints created when using them and the footprints created when not.
- Quality of the spatial relationships – Preservation of important spatial relationships of the actual with the approximated area e.g., if a data point is within the actual is it within the approximation.

Interestingly the first two are quantitative measures while the third is qualitative as described, although it is possible to see how it could be made quantitative by defining all the required spatial relationships and finding the percentage that are inconsistent between the approximate and the actual areas. This level of reasoned assessment is notably absent from much of the literature. Although this is undoubtedly because Alani *et al.* have an expected shape to measure against, it seems strange that little has been done to give any form of general scoring to the footprints produced.

α-shape, $\mathscr{A}$-shape and DSAM are all of complexity $O(n \log n)$. This is common amongst the footprint algorithms, at least in part because they tend to be generalisations or modifications of existing $O(n \log n)$ algorithms (e.g., Delaunay triangulations, Voronoi diagrams and Jarvis March). Aware of this, Chaudhuri *et al.* [14] propose two methods for extracting the 'perceptual border' of a dot pattern that have $O(n)$ complexity; the s-shape and the r-shape. The s-shape is generated by laying a grid over the isothetic (axis-aligned) bounding rectangle of the dot pattern. The union of all the grid cells that contain at least one dot gives the s-shape, s being the length of the grid cell sides. Chaudhuri *et al.* note that choosing s is not a simple task and the interesting component of the method is the fashion in which they deal with this problem. First they note that there are a finite number of different s-shapes that can be created for any dot pattern, by defining the sequence $\langle s \rangle$ as:

$$\overline{s} = \sqrt{\frac{A(W)}{n}} \quad \text{NB:}^{16}$$

$$s_i = \overline{s} \qquad \text{when } i = 1$$

$$s_i = \sqrt{\frac{A(H(s_{i-1}))}{n}} \qquad \text{when } i > 1$$

W: The minimum isothetic bounding box

$A(x)$: The area of the footprint x

$H(s)$: The footprint (hull) when the grid length is s

$\langle s \rangle$ finishes when each grid cell of the footprint contains only one point. This sequence gives rise to the 's-shape spectrum' of the dot pattern (similar to the work by Edelsbrunner for α-shapes) and can be done in $O(n)$ time. To choose an appropriate s value from the spectrum they introduce the parameter ε as a measure of disparity within the dot pattern, essentially how uniform the density is across the pattern. s is now chosen from the spectrum by:

$$s = \max \left\{ s_k; \left| \frac{s_{k-1} - s_{k+1}}{s_k} \right| \leq \varepsilon, s_{k-1} \epsilon \langle s \rangle \right\}$$

For the majority of patterns they found that an ε value of 0.3-0.5 was sufficient to achieve a suitable representation. It should be noted that they give little discussion on how the representation is measured and do point out that the 'perceptual structure' is not necessarily unique. The s-shape is staircase like and may be considered somewhat crude, whereas the r-shape is a much 'smoother' representation. The algorithm involves placing a disc of radius r over each dot and then joining edges between dots whose discs share a point on the boundary of the union of all of the discs. Like the s-shape it suffers from the difficulty of selecting a suitable value of r. Chaudhuri *et al.* observe this difficulty and proceed to show that using the s-shape algorithm a value can be retrieved for r where $r = \sqrt{2s_i}$, combined with the ε-measure of dispersion this gives a $O(n)$ method for producing a footprint with a single parameter and suggested ε value of 0.4. Chaudhuri *et*

[16] If the distribution of the dot pattern was completely uniform this would give an optimal value of s.

al. appear to have covered all the major issues (visual salience, complexity and parameter choice) in footprint generation but they do not make clear why they consider any r-shape is more suitable than any other or if it can be judged by anything other than human intervention.

There appears to be a division in types of footprint algorithms appearing, the α-shape, $\mathscr{A}$-shape and DSAM are all mathematically derived algorithms, they arise from the implementation of easily expressable concepts:

- α-shape – The intersection of all closed discs with radius $1/\alpha$ that contain all the points of S.
- $\mathscr{A}$-shape – The union of the cells containing dots of S from the Voronoi Diagram of $\mathscr{A} \cup S$.
- DSAM – The union of the cells containing dots of S from the Voronoi Diagram of $E \cup S$ where E is a set of dots known to be external to the query area.

However s-shape and r-shape are somewhat different; the basic description of the s-shape seem to be of the same type i.e.

- s-shape – The union of the grid cells of length s containing dots of the pattern.

But the s-shape algorithm, when the iterations used to generate the s-shape spectrum are included, consists of many more steps than this, in fact to properly describe the s-shape is to describe each of the steps taken within its algorithm. The same is true for the r-shape, particularly when taken with the s-shape as a precursor. While this thesis is not meant to be a detailed analysis of the types of algorithms available, it is interesting that there should be delineating characteristics between the algorithms. Further work in this field could entail examining which type of algorithms can be used to produce which types of footprint.

One of the few works to formally analyse the footprints created by their algorithms is the paper "What is the Region Occupied by a Set of Points?" by Galton and Duckham [28]. Galton and Duckham approach the concept of finding an appropriate footprint for a dot pattern by first looking at what was meant by the concept of 'appropriate'. Before examining the footprint criteria they point out that visual salience is problematic in that human intuition can play a great part in the shapes we see when we look at dot patterns, they note that the notion of gestalt perception almost certainly comes into play. Before describing their criteria for analysis they make one last caveat in that the specific application must decide the relevance of the footprint, by this they mean that whether or not the footprint is a suitable representation is dependent on the application. The nine general criteria they provide are, in fact, questions for which a specific algorithm should give answers in order to be compared to other algorithms to assess suitability for use in a specific application. These criteria are as follows (in which $H(S)$ is the footprint producing function H over the set S):

1. Should every member of S fall within $H(S)$ or are outliers permitted?

2. Should any points of S be allowed to fall on the boundary of $H(S)$ or must they all lie within its interior?

3. Should $H(S)$ be topologically regular or can it contain exposed point or line elements?

4. Should $H(S)$ be connected or can it have more than one component?

5. Should $H(S)$ be polygonal or can its boundary be curved?

6. Should $H(S)$ be simple, i.e., its boundary is a Jordan curve or can it have point connections?

7. How big is the largest circular (or other specified) subregion of $H(S)$ that contains no elements of S?

8. How easily can the method used be generalised to three (or more) dimensions?

9. What is the computational complexity of the algorithm?

The authors note that the criteria can be split into four categories. The questions (1) and (2) focus on the relationship between the footprint and the dot pattern. (3)–(6) describe the nature of the footprint itself. (7) is, in some respect, an indicator of the quality of the footprint, in that reducing the amount of 'free' space is important for a visually salient (this is exapanded on by Galton in [27]). (8) and (9) are both questions about the nature of the algorithm. They use these criteria to compare three algorithms and the general class of convex hull algorithms. The questions can be split into two categories, questions (1)–(7) are concerned with the state of the footprint whereas (8) and (9) describe aspects of the algorithm's nature. We note this as any convex hull algorithm will give identical answers to questions (1)–(7) differing only on (8) and (9). The three algorithms compared are the Swinging Arm, Close Pairs and a Delaunay triangulation based method.

The Delaunay triangulation method is extended into the χ-hull in [20] and is examined in greater detail later in this section. The Swinging Arm method generalises the 'gift-wrap' algorithm for constructing convex hulls[17], which is constructed by taking an extremal point p_0 of S and half-line l anchored to p, l is swung in a clockwise direction about p_0 till it collides with another point of S, p_1. l is swung successively from p_i to p_{i+1} till $p_{i+1} = p_0$. The Swinging Arm is identical save that instead of a half-line a line segment of length r is used. Interestingly this change allows that an anti-clockwise direction of spin can change the footprint produced.

The Close Pairs method considers simply joining all point-pairs whose distance is less than or equal to r, then taking the union of all the closed polygons as the footprint. With regard to how Swinging Arm and Close Pairs compare, the authors note that in most cases they are identical, save for criteria (8) and (9). So similiar that their produced footprints are identical save for on dot patterns for which the Swinging Arm would generate different results if the direction was changed.

[17]The 'gift-wrap' method is a renaming of the Jarvis March mentioned earlier.

The extensions into three dimensions is not particularly obvious for the Swinging Arm, the arm can easily be conceptually thought of as a 'flap', but the edge about which to rotate the flap is not pre-determined and would need to be decided on. Close Pairs generalises relatively easily, after including any polygon formed from the joins any polyhedrons with said polygons for borders are included.

The complexity of both of the algorithms is at least $O(n^2)$ with a worst case of $O(n^3)$ for Swinging Arm and an unknown worst case for Close Pairs. This kind of systematic comparison does not appear prior to this paper and will be looked at in greater detail in Chapter 4. For the moment we note that being able to compare the footprint types and their algorithms can be useful in the assessment of suitability for any specific application.

The final algorithm that will be examinee in this section is the χ-hull by Duckham *et al.* [20] (expanding on the Delaunay method presented in Galton *et al.* [28]). This paper includes a discussion of the footprint's properties, and how these are directly tied to the method by which it is created. The method itself is simple to understand; starting with the Delaunay triangulation and successively removing 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, however, like Chaudhuri *et al.* , examine the parameter choice and its effect. There are practical limits on the minumum length l for any triangulation, if it is too large then no lines will be removed and 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 further proof of the desirability of a non-parameterised algorithm.

As previously mentioned, within the field of footprint algorithm generation there is little in the way of hard analysis of the footprints, the algorithms in relation to each other or the patterns. Clearly such work is relevant to the field and much of what has been done has only been performed recently. In 2008 Galton wrote a paper [27], searching 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.

This work was followed by a paper by this author and Galton [21], suggesting a method for classifying the footprints. Unlike Galton [27] it does not look at their 'fitness' but approaches the subject from a desire to be able to describe algorithms by the types of

footprints they can create. The paper notes that the context in which the algorithm is being used determines the type of footprint that is satisfactory. With this in mind it proposes a method of using the application specific knowledge to limit the choice of algorithms for any particular user requirement. The classification bears some similarity to the set of criteria proposed by Galton and Duckham [28] for evaluating the footprints produced by different algorithms and will be detailed in Chapter 4

2.6 Dynamics

The focus of this thesis is not an examination of footprints or the underlying dot patterns but how the change of the phenomena can be suitably measured. It is therefore prudent to devote some attention to examining the existing approaches to dynamic updates.

The Kinetic Data Structures (KDS), proposed by Basch *et al.* [6], are a particularly appropriate starting point because one of the applications they use is that of maintaining convex hulls under movement. The KDS is not a single algorithm, rather it is a system to describe how to create dynamic algorithms. A set of conditions, called certificates, are defined. These certificates are geometric relations that describe the shape to be maintained such that if a certificate is not true (has failed) then the desired shape cannot exist. We can therefore ascertain whether or not the convex hull needs to be redrawn based purely on whether or not these certificates have failed. Obviously a KDS is only useful if the cost involved in discovering and processing certificate failure is small. They state that the cost is small if it asymptotically of the order of $O(\text{Polylog}(n))$, or $O(n^{\epsilon})$, for some small $\epsilon > 0$. A KDS with such small costs is deemed *responsive*. Futher to this a KDS is *efficient* if there are very few internal events compared to external events[18], *compact* if it has a near linear number of certificates and *local* if no object participates in too many certificates. The change identifiers can be treated as certificates on the dot pattern, and a framework using them as a form of a KDS. A change identifier KDS should be responsive and, ideally, compact. The terms efficiency and local have no obvious counterpart in a change identifier based KDS as they are specific to certificates on the footprint.

The KDS uses short term motion plans for the objects, these are used to sort the events into queues such that the most likely certificate failures (events) are looked at first. An example on convex hulls is given where it is shown that by checking a set of certificates, all using the rule $\text{ccw}(a, b, c)$ in which a, b and c are points and the relation is true if they form a counter-clockwise triangle, any events whereby the convex hull is no longer valid for the dot pattern can be found. Further to this they provide a more robust method using the dualities of the convex hull and focusing only on the upper envelope. All of this gives an excellent base from which to work but it may not be a directly transferrable approach for footprints. This is because, unlike convex hulls and as discussed earlier, footprints are vaguely defined. As a result, choosing the certificates is no longer a trivial task. We will look at possible requirements that can be made on footprints in Chapter 4.

[18] *External Event*: Changes the shape. *Internal Event*: Shape stays the same, certificates change.

Hershberger and Suri [36] have a very different idea to Basch *et al.* using a technique called adaptive sampling. By using an approximation of the extrema from the dot set they find a convex hull that approximates the 'true' convex hull, with triangles of uncertainty (see Fig. 2.6) over each line segment. The area of uncertainty is given by the supporting line perpendicular to the direction in which the point was found.

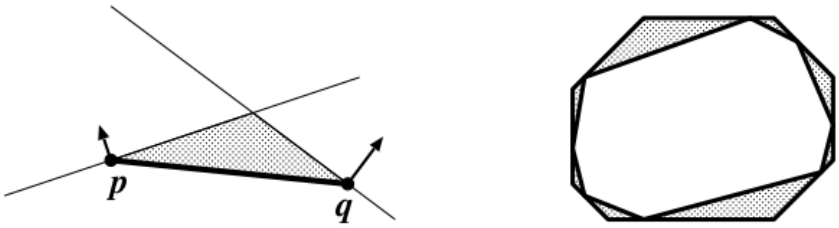


Figure 2.9 Example of Uncertainty Triangles. Image from [36]

While sampling is not a new concept Hershberger and Suri suggest that uniform sampling produces poor quality approximations in low curvature regions. As such they propose an adaptive sampling scheme.

The method is performed by first uniformly sampling extrema in directions $2\pi/r$ for $j = 0, ..., r-1$ then adding up to r more extrema using their adaptive technique. Given an edge e, $\Theta(e)$ is defined as the minimum angle between the directions the endpoints e were sampled in. $\Theta(e)$ and the proportion of the perimeter that the length of e represents are used to provide a *sample weight* $w(e)$ for e. If $w(e)$ is greater than one then the edge is refined; the extreme point is found in the direction that bisects the angular range defined by e's endpoints. If the point found is not an endpoint of e then e is replaced by the two new edges of the vertices of e and the newly found point. This greatly reduces the error in the approximation. As has previously been discussed, the approximation of a footprint is central to the concept of the use of change identifiers.

Chiang and Tamassia [15] present a general review of the field. While it is a little dated (1992) it serves as a good presentation of methods still in use. Unlike Basch *et al.* or Hershberger and Suri, Chiang and Tamassia look at the ways in which the data structure holding the dynamic dot pattern and the footprint is maintained. Chiang and Tamassia examine various forms of binary trees and *fractional cascading*. Using these data structures they approach some general dynamic methods. The examination is quite extensive so we have picked some terms which may be considered when choosing the data structure(s) used to contain the dot patterns.

- **Local rebuilding / Balancing** This is a technique applied to search trees so that they maintain logarithmic height.
- **Partial rebuilding** This rebuilds entire subtrees when they become out of balance.
- **Global Rebuilding** Periodically reconstructs an entire tree, often used with 'weak'

updates (like lazy deletion).

- **Lazy Deletion** Does not remove deleted item but marks it as deleted to be dealt with during the reconstruction.

- **Decomposable** A search problem is decomposable 'if for any partition (S', S'') of S the answer to a query on S can be obtained in constant time from the answers to queries in S' and S''.

Further to the discussion on general dynamic methods considerations Chiang and Tamassia give a list of things we can reasonably expect from any dynamic algorithm for convex hulls:

- find if a given point of S is on the convex hull H of S;
- find if a query point is internal or external to the convex hull H of S;
- find the tangents to the convex hull H of S from an external query point;
- find the intersection of the convex hull H of S with a given query line;
- report the points on the convex hull H of S.

They note that the set of points S is updated only by insertions and deletions, so any point movement should be treated as being removed then added as a new point which, fits with our above-mentioned concerns about how the data may enter the framework. Chiang and Tamassia describe a method by Preparata with update and query time of $O(\log v)$ and a report-query time of $O(v)$, where v is the number of vertices currently in the convex hull H of S. The method only deals with insertions and is therefore not entirely applicable to change identifiers but it does introduce the concept of splitting the footprint into an upper and lower hull, the methods used for the upper are transferable to the lower. This concept seems common, appearing in the next algorithm and in the KDS example.

Overmars and Leeuwen [52] present an algorithm for processing what they call fully dynamic hulls which can handle both insertion and deletion. Again this considers splitting the hull into two sets, one for left and one for right. Splitting the hull is a useful technique for speeding up computation but it requires that the two sides are comparable. Footprints considered more generally than the convex hull do not necessarily have the strict mathematical definition that allows us to make such a split.

As with all the work described in this chapter the above discussion of dynamics is not a complete listing. There is, for example, the work performed by Gold [30] examining the nature of multiple dimensions for dynamic data structures. The field is extensive but is not exhaustively detailed here as the other works do not directly impact on this thesis.

2.7 Summary

The papers we have examined here are by no means the sum of all the papers in the fields represented. We have instead intended to give an overview that shows the related materials and, most importantly, the methods that have been used to analyse dynamic dot patterns, footprints and dynamic footprint assignment.

With regard to the originality of our work, there has been much work on the field of spatio-temporal data and dynamic updates of convex hull. None of the papers that were found in researching for this thesis has given the impression of being of the same nature as our change identifiers. While the change identifiers do have similarities with some existing ideas, the examination of the spatial properties of the dot pattern as a static abstraction combined with the allowance for some 'error' when updating footprints (considering any footprint can be seen as an approximation of some abstract 'true' or 'best' region for the dot pattern) makes them a unique concept.

3 Dot Patterns

The previous chapter was an examination of existing literature that included work that has constructs with the same or related structures to that of dots, dot patterns and dynamic dot patterns. The definitions of the structures this thesis uses were given in the introduction (Chapter 1) and this chapter will expand upon these definitions in light of the existing work. In particular we examine the descriptors that can arise from looking at the dot pattern as a static representation.

3.1 Change and Dot Patterns

The abstraction provided by considering the entities as dots means that change as it relates to a dot can only occur in three ways: movement, disappearing or appearing [58, 38]. These simple actions can lead to complex emergent behaviour in the dynamic dot pattern; for example expansion, dispersion, rotation, etc. These behaviours can be measured by looking at the change in the properties of the patterns between two adjacent (as defined by Stell [57]) phases of a dynamic dot pattern. Expansion, for example, is a measure of increase in extent between two phases and can be determined by the positive change in the standard deviation from the centroid. There is more than one method by which to measure extent (standard deviation, bounding box area, etc.) and change in any of these properties indicates a value for the complex behaviour of expansion (negative or positive). As mentioned in the introduction, these different properties are *descriptors* of a pattern while a general property characterisation such as 'extent' is a *class* of descriptor.

Using this reasoning a definition of change for collectives represented by dynamic dot patterns can be given by the difference between two phases in one or more descriptor classes measured by descriptor methods from those classes.

3.2 Descriptor Classes

For the measurements of change to be useful across any given dynamic pattern we need to be able to state, with some confidence, that any of the complex behaviours a collective represented by a dynamic pattern can exhibit will be 'caught'. The framework this thesis presents should not have a list of exception cases in which it cannot accurately identify change, e.g., 'the change identifiers will not correctly cause a footprint update if the dynamic dot pattern rotates'. There are numerous, possibly incalculable, different

descriptors for a dot pattern; any statistical, geometric or otherwise calculated value that can be assigned to a pattern is a valid descriptor. Using all possible descriptors is not only a Sisyphean task, it will also probably involve repeated measurements within the same descriptor classes. It follows that we should instead focus on having a descriptor method from each class.

The classes are not, however, a strictly delineating classification – some descriptors straddle the boundaries of multiple classes and some classes are clearly dependent on others; as such the class divisions should be used as a guide rather than a set of requirements. Choosing the classes is not a simple task, there are multiple aspects of the dots which can be measured and the classes selected must be those that could be considered both broad and descriptive. To deduce the appropriate classes from first principles the most basic information of the pattern is investigated, the exploration increases in complexity until we have covered a sufficient range of aspects. The cardinality of a dot pattern is, perhaps, the most immediately apparent datum of information that can be attained. However, there are no alternative methods to counting with which to find the cardinality of a pattern, and cardinality is therefore not so much a class as it is a descriptor.

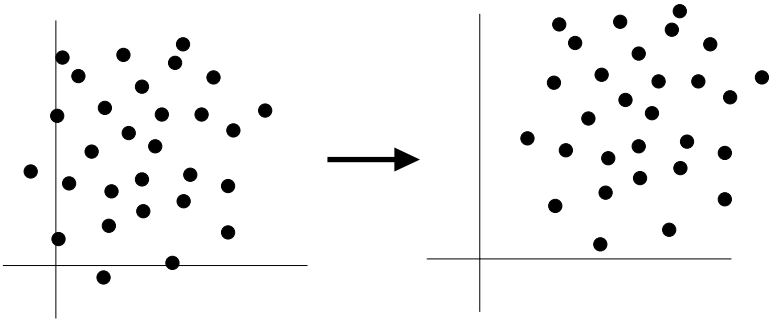


Figure 3.1 Dynamic dot pattern changing in position.

The dots are little more than a location and it seems sensible that the first aspect of a pattern we would need to be able to describe is its *position* (Fig. 3.1). Position can be measured in multiple ways but will tend to return a vector as its value, usually a coordinate location. This observation is important as such multi-part values can lead to issues when normalising. For the location change to be proportionate we need to have some value by which to make it so. If a pattern with an areal coverage of 1000 units^2 moves by 1 unit very little change has occurred, if however the areal coverage is only 4 units^2 such a change is quite large (an example of this is shown in Fig. 3.2[1]). Position, then, leads directly to our next class which describes the size of the pattern; so as not to be confused with the cardinality we shall name this class *extent* (Fig. 3.3).

A pattern's extent and location have been considered, if we require that we must be able to measure change in at least all the standard affine transformations (i.e., translation, scaling rotation and shearing), and that rotation can occur without change in extent or location, then the third class should be *orientation* (Fig. 3.5). Shearing a pattern will change the orientation of a pattern so does not need to be directly measured; this can be seen demonstrated in Fig. 3.4.

[1] Although the areal coverage values are not the same because of space constraints.

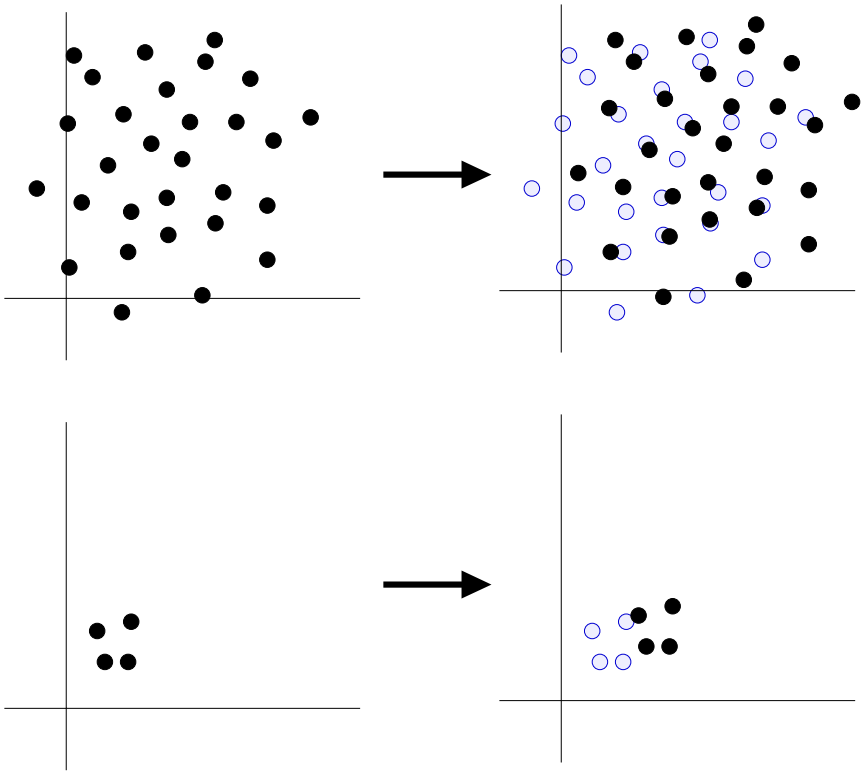


Figure 3.2 Demonstrating the relative difference in position change of two patterns of different extents.

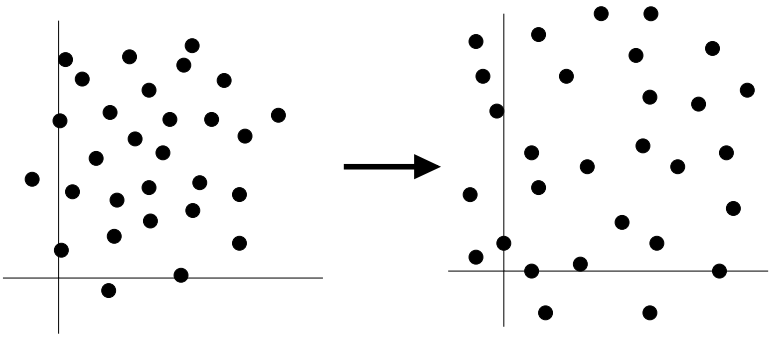


Figure 3.3 Dynamic dot pattern changing in extent.

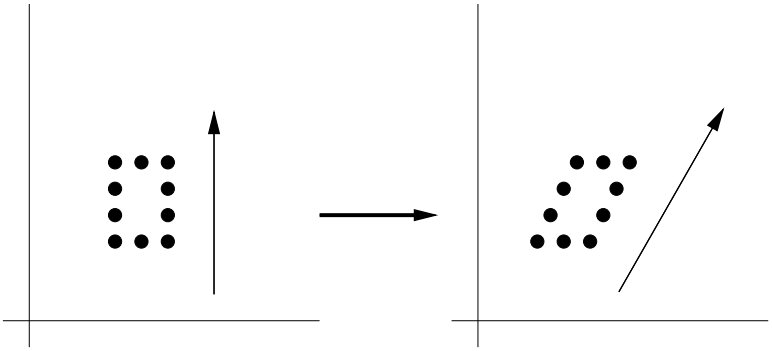


Figure 3.4 Dynamic dot pattern shearing, the arrow represents the pattern's orientation.

The three classes described above (*position*, *extent* and *orientation*) treat the pattern as an region with its own properties, as opposed to a set with no properties beyond its elements. Further traits a region can have are given by the shape descriptions given by Galton [26]

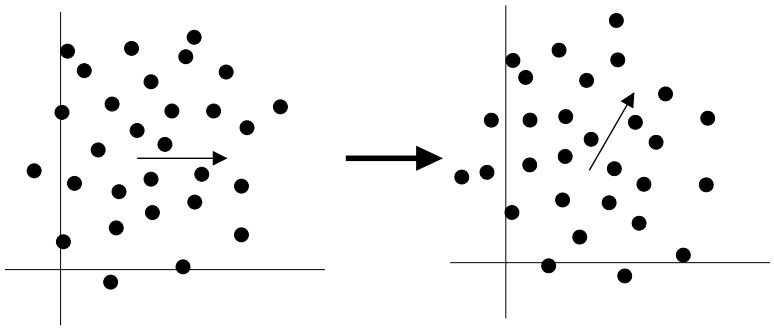


Figure 3.5 Dynamic dot pattern changing in orientation.

that were detailed in the background chapter. The two which have not yet been considered are *connectedness* and *dimension*. As was previously noted, Galton's shape descriptions of connectedness makes little sense when applied to dot patterns as the dots are by their nature disconnected. However it is often the case that patterns can appear to have areas of differing densities, which could easily be seen as disconnected components of a pattern (Fig. 3.6). Measuring connectedness in this fashion is not straightforward; do we consider the degree of connectedness or do we measure the number of distinct components? If we measure the number of components how do we place a threshold on when we consider a disconnection to have occurred? Connectedness cannot be ignored as too problematic a class, as it is indicative of complex underlying behaviour of the type described by Huang *et al.* [38] (merge and split) and we will look at how it may be measured when methods for the descriptors are introduced.

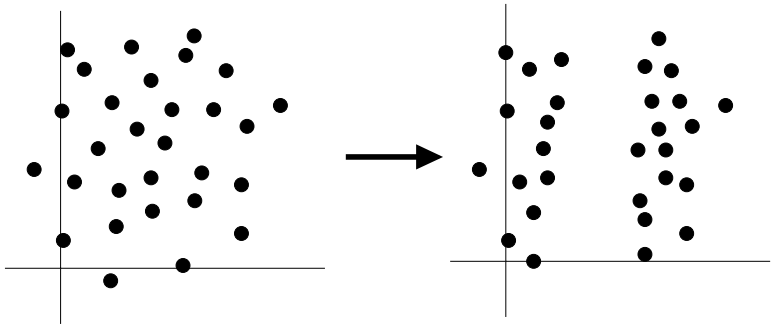


Figure 3.6 Dynamic dot pattern changing in connectedness.

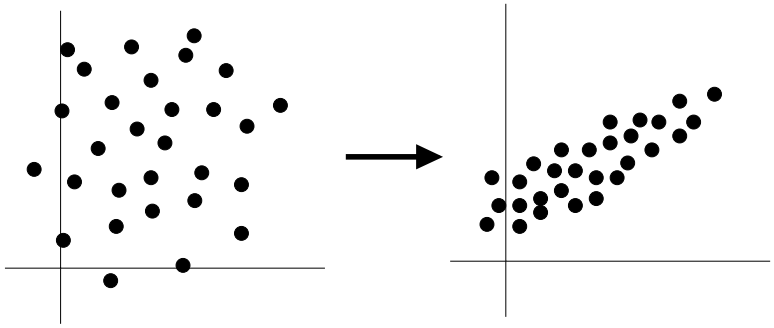


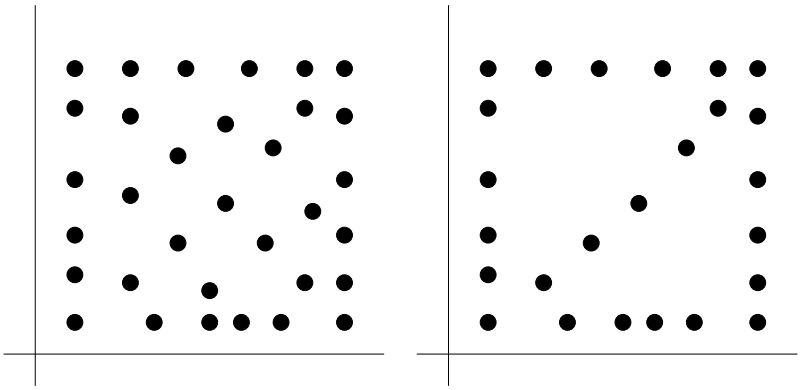
Figure 3.7 Dynamic dot pattern changing in dimension.

As discussed by Galton [26, ch. 4.7.3] Dimension is split into two approaches; strict and apparent. As was mentioned in the background (Chapter 2), strict dimension is not

particularly useful for describing dot patterns; it is always possible to draw a curve through all the dots in a pattern and thereby call it 1-dimensional. We are more interested in apparent dimension; the idea that a 2-d pattern can be close enough to linear as to, at a coarse enough granularity, appear 1-dimensional (Fig. 3.7). Change in dimension may not be picked up by identifiers for the previously described classes; thus dimension is the next identifier class. Like connectedness, dimensionality requires a choice between either stating to what degree a pattern is in the dimensions of the space in which it is embedded or measuring the integer value for which dimension appears to represent it. This can be solved by having multiple descriptors; measuring the degree of dimensionality for each dimension the pattern can exhibit and comparing their values. For 2-dimensionally embedded patterns this is not necessary because the pattern can only exhibit apparent dimensionality in 0 or 1 dimensions. An apparent dimensionality of 0 (the pattern is densely clustered around a single point) will be identified by extent descriptors; leaving only a measure of apparent dimensionality 1, or collinearity, to be implemented.

With extent, orientation, position, connectedness and dimensionality change in the simple affine transformations and in the herd evolvements of Huang *et al.* [38] can be tracked.

The current classes all treat the pattern as if it had a surrogate region, so changes which may affect the footprint arising from its collective nature should also be considered. As an example of the distinction consider the two dot patterns shown in Fig. 3.8.



(a) Pattern DP_x (b) Pattern DP_y

Figure 3.8 Two Dot Patterns with the Same Extent, Position, Orientation, Connectedness and Dimensionality

Depending on the methods used as descriptors both DP_x (Fig. 3.8(a)) and DP_y (Fig. 3.8(b))can have the same extent, position, orientation, connectedness and dimensionality. However DP_y has large empty spaces. Measuring cardinality would prevent this but we have already discussed why it is not sufficient as a class of descriptor. Instead we track the *distribution* of the dots within the pattern, this allows descriptors that measure homogenity, global density and cardinality to be tracked, all of which are likely to show different measures for DP_x and DP_y.

This section has presented list of descriptor classes that allow us to be sure that, if we have a descriptor for each class, we can cover the aspects of change that a dot pattern can

undergo. Consequently we can be confident that change identifiers will be able to update the footprint at appropriate change points.

3.3 Descriptors

With the classes identified we can begin to examine the actual descriptors. Those described below are by no means a complete list but instead give an overview of the descriptors used within this thesis and those that have been specifically avoided.

3.3.1 Position

To compute a value for position requires a point or space relative to which we can measure it. Even with a clear origin or frame of reference there are a range of different units of measurement for position, either with numerical or with qualitative values; e.g., polar or Cartesian coordinates and compass positions respectively. It makes intuitive sense to use the frame of reference in which the dots themselves are positioned and to use the same unit. However it is not clear how change would be measured for qualitative units without defining quantitative thresholds on them. To avoid the complication of further thresholding we will focus on the numerical measures.

Example Methods:

- **Centroid** – The mean average location of all the dots within the pattern.
- **Isothetic Bounding Box Centre** – The centre point of the axis-aligned (isothetic) minimum bounding box of the pattern.
- **Bounding Box Centre** – The centre point of the minimum bounding box of the pattern (non-axis aligned).
- **Minimum Disc Centre** – The centre point of the minimum bounding disc of the pattern.

It is apparent that there is a difference between the first and the last three given descriptors. The former descriptor treats the pattern as a set of points and the latter all apply a *surrogate* footprint to the pattern. Many of the classes have descriptors of each type. The minimum disc and the minimum bounding box (not axis aligned) are more computationally complex than the isothetic minimum bounding box and as a result may be unusable as change identifiers; it not being clear if they actually provide a sufficiently 'better' (more accurate) centre than any other measure for the greater computation time they take.

3.3.2 Extent

Unlike position, extent measures tend to be represented by a single value, but this value is not always of the same unit. Some descriptors are linear (e.g., pattern diameter) and others are of the order of the space the pattern inhabits (e.g., area of the bounding box). As we are currently only concerned with measuring the properties of the pattern, the relevance of unit type difference is not yet obvious, however when change identifiers are examined in Chapter 5 the unit type will affect how the values of change are normalised.

Example Methods:

- **Variance from Centroid** – The variance from the mean centroid. We use variance so as to avoid the square roots required by the standard deviation.
- **Bounding Box Area** – The area of the isothetic minimum bounding box.
- **Diameter** – The diameter of the pattern is the distance between the two furthest dots.

The diameter of the pattern is found by locating all the external dots i.e., all the dots that are vertices on the convex hull of the pattern. This may make it too complex to be used when change identifiers are considered, despite its conceptual simplicity. However an approximation can be found by considering the mean between the lengths of the diagonal of the isothetic minimum bounding box[2] and the longest edge of the isothetic bounding box[3]. Alternatively a less accurate estimation can be attained using the greatest distance between the dots in the extremal axis aligned dimensions (for a 2-dimensional embedded pattern this would be the dots with the greatest and least x and y values). The bounding box of a pattern requires the extremal dots to be found before computing the vertices of its corners and will therefore take longer to compute than the less accurate estimation. This thesis makes use of the axis-aligned extremal dots distance as its estimated diameter to provide a very computationally fast extent descriptor. To further increase its speed it uses the squared distance instead of the actual distance, preventing the computationally difficult task of the square root. Of note is that all three measurements return a squared unit and this should be considered when comparing them to other descriptors.

3.3.3 Orientation

Orientation is the direction in which the pattern is facing. As the dots do not have an associated direction, unlike position, this cannot be an aggregrate of individual values. In fact given the information inherent within the dot pattern a true orientation is impossible as even the 'line of best fit' will not tell you in which direction along the line that the pattern points. From a change identifier point of view this is irrelevant, as all we need is a measure which will change as the orientation changes; the measure does not need to describe the orientation exactly, as long as it is linked to it.

[2]Guaranteed to be no less than the length of the diameter
[3]Guaranteed to be no greater than the length of the diameter

Example Methods:

- **Gradient of Line of best fit** – There are multiple ways of measuring the line of best fit; however, the ordinary least squares (OLS) method is perhaps the least complex. OLS is a linear regression approach from the field of statistical analysis that minimises the squared vertical distance between a dot from the pattern and the line of best fit. As OLS is a statistical analysis technique it has some properties which do not apply directly to the use of spatial data. Within statistical analysis one of the variables would be expected to be an observed result dependent on another. For example when measuring the growth of children between the ages of 10 and 14, the height is observed data that is dependent on the age. Within a dot pattern all the variables[4] are independent and, as a result, we do not to take into account error in the observation. With no errors to concern us we can use the simplest form of OLS estimation to find the line $y = \alpha + \beta x$ in which:

$$\beta = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$
$$\alpha = \bar{y} - \beta\,\bar{x}$$

- **Gradient of the Principal Component** – Found by principal component analysis (PCA). PCA finds the dimensions (components) with the highest variability within the pattern and is commonly used as a dimensionality reduction technique. Formally it transforms the coordinate system the data resides so that, when the data is projected onto it, the distribution across first coordinate has the greatest variance, the second coordinate has the second greatest, etc. To find the Principal Component we find the eigenvector corresponding to the largest eigenvalue of the covariance matrix of the data. The principal component is not always the same as the line of best fit as the OLS method minimises only the distance in the y-axis (Fig. 3.9). Both PCA and the OLS linear regression technique are explained in more detail in [11].

3.3.4 Connectedness

Connectedness is actually a form of distribution measure as its measurements are performed by comparing inter-dot differences. It is, however, one that appears salient enough to warrant its own class. The patterns can often appear to split into separate groups and identifying the change in these groupings is similar to the behavioural evolvements of Huang [38] (herd splitting and joining). Connectedness in this fashion can be a discrete measure or a continuous value: the number of distinct groups and how connected the pattern is respectively. The continuous value approach, perhaps, fits better in the distribution class. Thus for the connectedness class we look only at the approach that provides an integer value for the number of clusters.

Example Methods:

[4] In a 2-dimensional pattern these would be the x and y coordinates

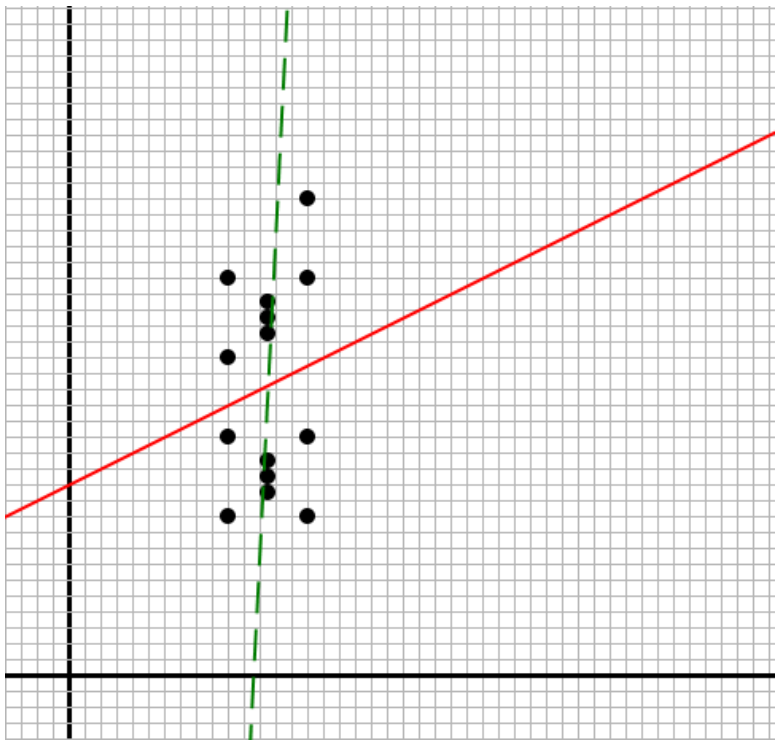
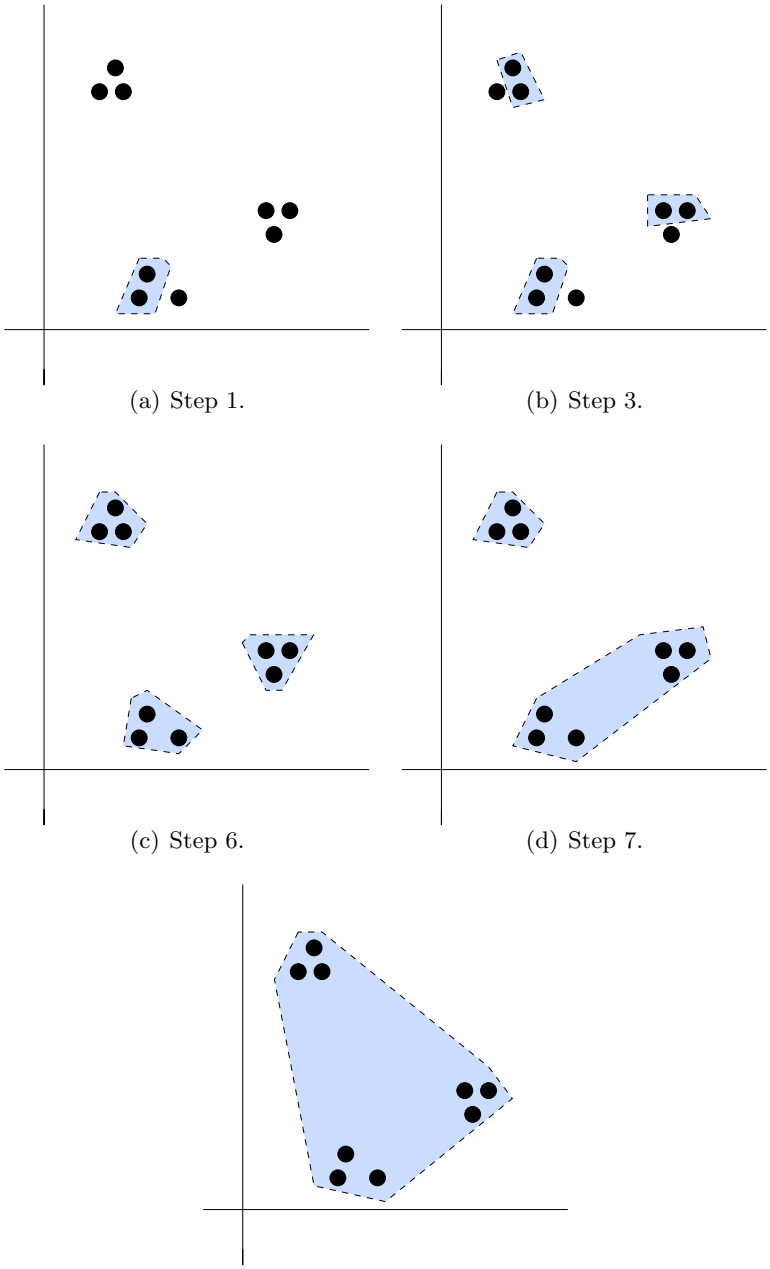


Figure 3.9 OLS (Solid line) VS. PCA (Dashed Line)

- **Greatest Jump Agglomerative Clustering** – This is an extension of the agglomerative clustering approach created for this thesis to give a value for the possible number of clusters. By running an agglomerative clustering method using, Euclidean distance as its metric, a hierarchy of possible clusterings can be created. Agglomerative clustering iteratively concatenates the dots into clusters by finding the closest distance between any two clusters. Fig. 3.10 shows an example of the agglomerative clustering process. We take note of the first unusually large jump in distance across the run of the clustering. If the jump is greater (by, for example, a multiplication of 2) than the average distance jump then the clustering preceding this jump is likely to contain a saliently identifiable set of clusters. In the example in Fig. 3.10 this would be the jump from step 6 to step 7 (Fig. 3.10(c) to Fig. 3.10(d)). This method is slow but effective at finding the parts of the pattern that we may identify as individual components.

- **K-Means Clustering** – An alternative to the *hierarchical* approach given by the agglomerative clustering method is the K-Means approach. Given a number of clusters K this minimises the squared distance between each dot and its closest cluster centroid. A full description of this method can be found in [11] but we assert that it is impractical to use it as a descriptor. While faster than an agglomerative approach there is no easy way of scoring the clustering it produces as to how well the it fits the data; this means that when iterating through possible values of K there is no definitive point at which to stop. We have tried several approaches, including balancing the number of clusters against the variance from the centroid for each cluster and minimising the nearest neighbour distance for each cluster, and have found the results to be less than satisfactory.



(a) Step 1. (b) Step 3.

(c) Step 6. (d) Step 7.

(e) Step 8.

Figure 3.10 Agglomerative Clustering Method

The agglomerative clustering method has to perform a large number of computations as it requires the nearest neighbours for each dot to be found (not estimated nearest neighbours as we will consider later) and involves an iterative process comparing cluster distances. As a result it may be infeasible to use as the basis for a change identifier.

3.3.5 Dimensionality

Measuring the appropriate dimensionality can be performed in several ways and, like connectedness, we can envision both discrete and continuous measures: measuring the apparent dimension the pattern is in and measuring the degree to which a pattern fits a given dimension respectively. Within this work all these measures are assuming a 2-dimensional

space so the dimensionality measures are focused on identifying how 1-dimensional a pattern is. However most of these are directly extendable into further dimensions.

Example Methods:

- **Correlation Co-efficient** – Measuring how closely the pattern conforms to its linear estimator, essentially how 1-dimensional the pattern is. The correlation co-efficient can be found in two ways. The first is sample correlation co-efficient, if s_x and s_y are the sample standard deviations of x and y respectively then the sample correlation co-efficient can be found by:

$$\frac{\sum_{i=0}^{n}(x_i - \bar{X})(y_i - \bar{Y})}{(n-1)s_x s_y}$$

  The second is the Pearson co-efficient correlation, if $\text{cov}(X, Y)$ is the covariance of X and Y and σ_X is the actual standard deviation (as opposed to the sample standard deviation used above) then the Pearson co-efficient correlation can be found by:

$$\frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}$$

  If we are using PCA to find the gradient then we already have the covariance matrix for X and Y so may be able to save processing time by using them in conjunction.

- **Principal Component Eigenvalue Difference** – Taking the pattern's eigenvalues and finding how weighted one is over the other. In 2-dimensions we can measure how 1-dimensional the pattern is by the difference between the principal eigenvalue and the orthogonal eigenvalue divided by their sum. The closer to 1 this value returns the more 1-dimensional the pattern.

3.3.6 Dispersion

This is possibly the most far-reaching of the classes in that it attempts to describe the layout of the pattern: How dense is it? How homogenous is its density? etc. As a result it has a large number of potential descriptors with a range of different complexities. It may be the case that multiple descriptors from this class are required to accurately measure change. If this is the case the class may need to be split into separate sub-classes[5].
Example Methods:

- **Cardinality** – Simply the number of dots in the pattern.

- **Global Density** – The global density of the pattern: Cardinality divided by an extent measure, usually the bounding box area but variance is an equally valid option.

[5]This relies on such classes being identifiable

- **Estimated Nearest Neighbour Distance Variance** – This is a method that returns a value for how clustered a pattern is and is computed as the variance in distances between estimated nearest neighbours. To avoid the high computational complexities suffered by finding nearest neighbours we use the nearest neighbour in the plane aligned dimensions. If the patterns are stored in a data structure that is sorted by all dimensions of the space the pattern is embedded in (e.g., in a 2-dimensional space a structure sorted by x and y) then this can be estimated in $O(\log n)$ time (assuming a $O(\log n)$ search time). It is not possible to be sure that the actual nearest neighbour has been found, as is demonstrated in Fig. 3.11. Using the estimated nearest neighbour difference, b would be identified as the nearest dot to O while a is its actual closest dot; the dots that surround O in the cardinal directions are closer in x or y coordinates than a and would therefore be adjacent to O in the data structure. For large dot patterns such exceptions will not greatly change the nearest neighbour distance variance so we can accept the estimation that this method produces.

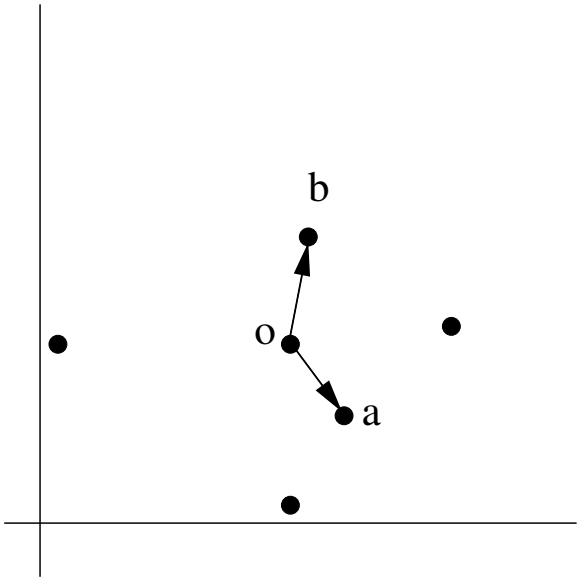


Figure 3.11 The case when the x and y ordered trees will not locate the nearest neighbour. $\vec{oa}$ is 5 units in length while $\vec{ob}$ is 6.1 units.

- **Skewness** – Skewness is a measure of the how uneven the distribution of a pattern is from the mean, often thought of as how much the histogram of the data slants to the left or right. It is given by:

$$s = \frac{\mu_3}{\sigma^3}$$
$$\text{where } \mu_3 = \mathrm{E}[(X - E[X])^3]$$

  In which σ is the standard deviation and $E[X]$ is the expected value of X, for the purposes of a dot pattern this is equivalent to $\bar{X}$.

- **Kurtosis** – Kurtosis is the measure of how even the distribution of the pattern is, often thought of as how flat its histogram would be. Kurtosis makes use of the fourth

moment about the mean[6] and is therefore related to skewness which uses the third. It has the equation:

$$k = \frac{\mu_4}{\sigma^4}$$
$$\text{where } \mu_4 = \mathrm{E}[(X - E[X])^4]$$

3.4 Descriptor Analysis

The descriptors range from simple measures to complex statistical analysis. For the purposes of this thesis it is necessary for the descriptors, when used within a change identifier framework, to be computable within a reasonably short time. At this point we must clarify that the descriptors are not algorithms, although descriptors often suggest an algorithm. For example, Kurtosis is strictly defined by a mathematical formula but there are numerous ways to implement the formula. It is possible that the more complex descriptors will fail to meet this time constraint (e.g., greatest jump agglomerative clustering), particularly considering that multiple descriptors will likely need to be used to fully explain each pattern. However, it is also known that many of the descriptors will operate within the same classes and, as described above, therefore only a subset of the descriptors may be required. The task is to decide upon a set of descriptors for which each descriptor provides unique information while minimising the time it takes to run the set.

To assess information redundancy a method is required to identify the similarities between the descriptors; using heatmaps of a correlation matrix provides a visualisation of the relationships between descriptors. A heat map [55] is a graph for which each mapped point is shown a colour on a spectrum, with high values at one end and low values at the other. The heat map shown below uses the standard approach of 'cold to hot' colouring, with low negative values being blue and high positive values being red. Given that the map is overlaying a correlation matrix, red indicates a strong positive correlation and blue indicates a strong negative correlation. Before creating the correlation matrix a set of dot patterns on which to run the descriptors is required. For a dynamic dot pattern based on a collective each pattern is related to the others within the set; usually as the dots have a concurrent identity, representing the same entities. This relation means that a correlation matrix of descriptors over a dynamic dot pattern will likely show false correlation. For example if the dynamic dot pattern represents a migrating herd: The herd is traversing a particularly wide and long stretch of their route leaving themselves open to predation. As a result the herd's extent decreases as the animals huddle together for safety, nevertheless some of the members are caught and devoured. Such a dynamic dot pattern will show strong correlation between cardinality and bounding box area. While these descriptors may well be correlated there is no way of knowing if the correlation is indicative of the behaviour of the underlying collective or the relation between the descriptors. To make sure that such possible false assumptions are avoided an automated

[6] Moments about the mean are values used to describe probability distributions and have the general form $\mu_k = \mathrm{E}[(X - E[X])^k]$ for the k^{th} moment about the mean, for example the second moment about the mean is the variance.

method is used to generate sets of unrelated and randomly shaped patterns (while being aware of pitfalls of assumptions based on test data described by Laube [46] and discussed in the background chapter Chapter 2). The randomly generated patterns can contain obvious concavities and can differ in extent, cardinality, connectedness, dimensionality, distribution and orientation. An example of some of the patterns that can be produced by this method is shown in Fig. 3.12

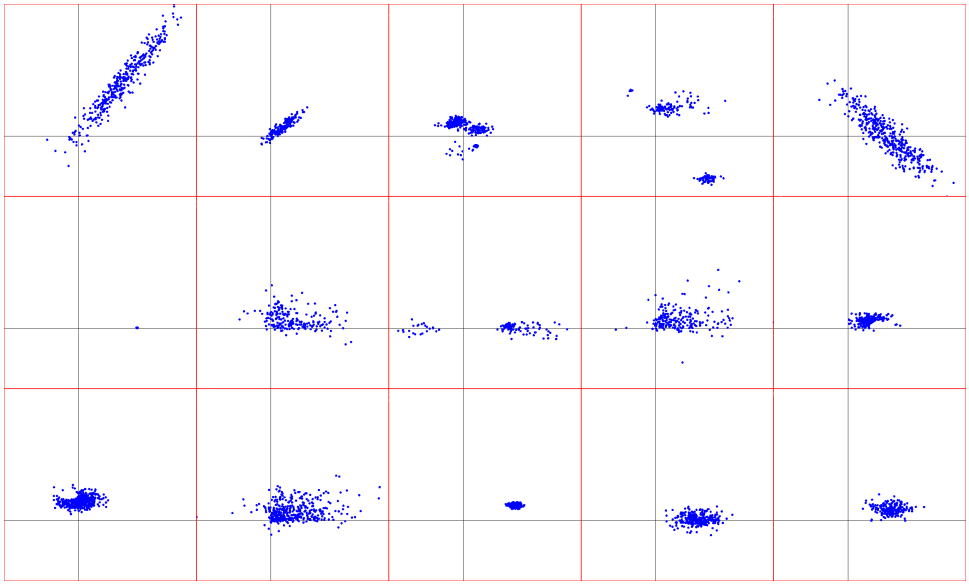


Figure 3.12 Examples of dot patterns produced using the random generation method

We must be careful, however, not to assume classhood solely on the basis of a correlation of heatmap values. For example the first heatmaps produced showed a correlation between the variance of nearest neighbour distance and the area of the bounding box. This is a logical result as the smaller the area in which the dots are situated the more likely they are to lie close to each other. However this correlation is not one that shows redundant information as the *aim* of the nearest neighbour distance variance is to describe the dispersion of the pattern whereas the area intends to show extent. Instead of finding a correlation indicating concurrent classhood instead a flaw has been found in the nearest neighbour distance variance. To remove this false concurrency the nearest neighbourhood variance needs to be scaled by an extent measure; the important information is not the variance of the distance but the variance in what proportion of the available distance is taken up by a dot and its nearest neighbour. The heatmaps, therefore, do not show just which descriptors are superfluous but also those which need to be modified before use.

Position is not included within the heatmap as it is, for 2-dimensional patterns upwards, at least a pair of values and as such cannot be used within the correlation matrix. However even if we could include position within the heatmap it is unlikely to provide any correlations; all the descriptors that use positions[7] apart from the eigenvalue difference are relative to the centroid.

Fig. 3.13 shows a map for all the descriptors together. We are primarily interested in uniqueness within each class but by graphing them all together we may find unexpected

[7]Of the descriptors we provide, only cardinality does not make use of position data

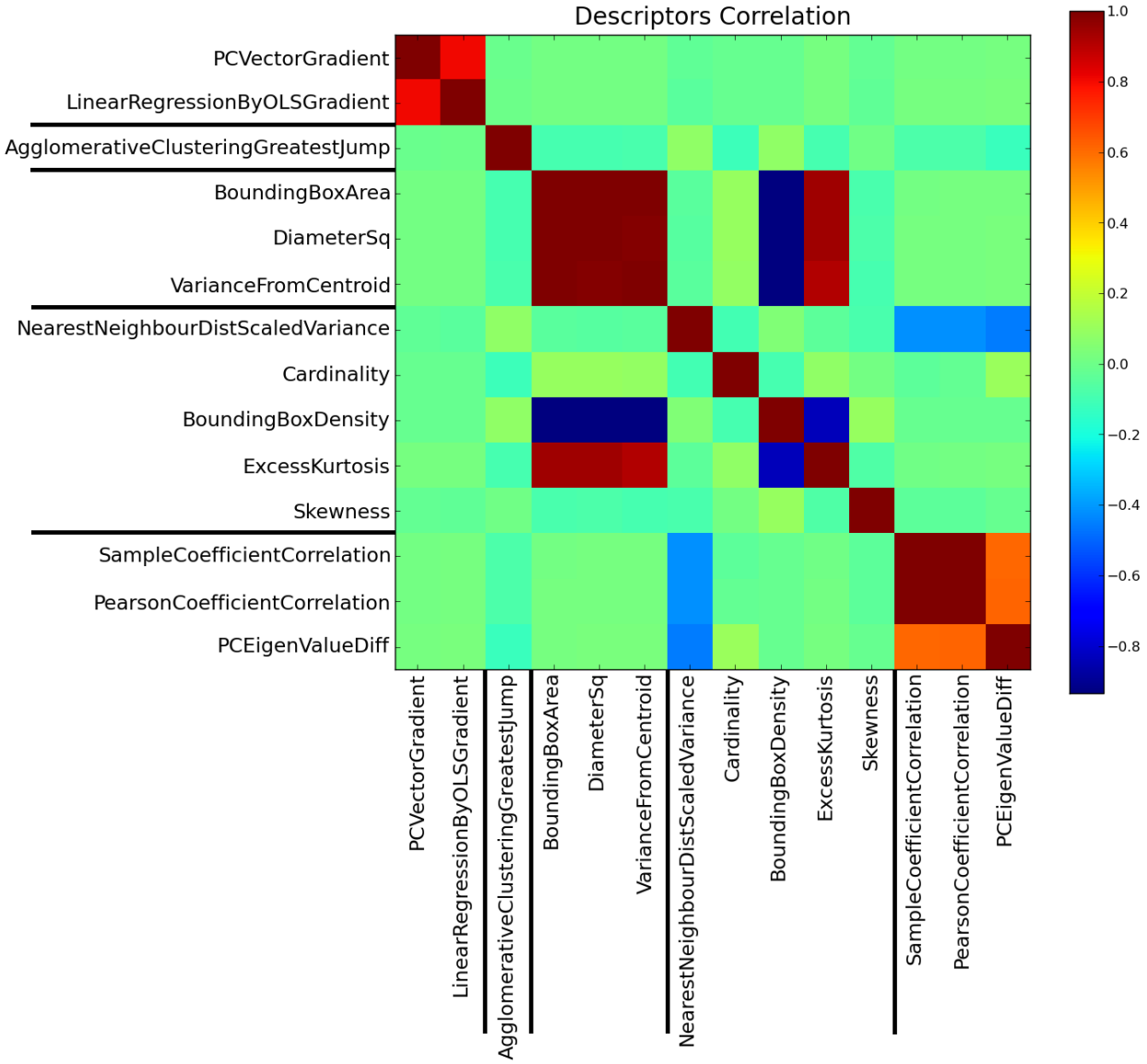


Figure 3.13 Correlation Heat Map: All Descriptors

correlations. The descriptors have been grouped by their respective classes:

Orientation: Principal component vector gradient and gradient of 'Line of best-fit' found via the linear regression method of Ordinary Least Squares.

Connectedness: The number of clusters with in an agglomerative clustering heirarchy that is found at the stage before the greatest jump in distance when joining the clusters together.

Extent: Area of the bounding box, the estimated diameter squared of the pattern and the variance from centroid.

Dispersion: The variance in distance between nearest neighbours, the cardinality of the pattern, the density of the pattern within its bounding box, the excess kurtosis and skewness.

Dimensionality: The absolute sample coefficient correlation, the absolute Pearson

coefficient correlation and the absolute difference between the eigenvalues divided by their sum.

While above it was shown that the orientation methods can produce very different results (Fig. 3.9) the heatmap demonstrates that they increase and decrease in their values concurrently; leaving the choice of the most appropriate descriptor to be based on which can be performed in the fastest time.

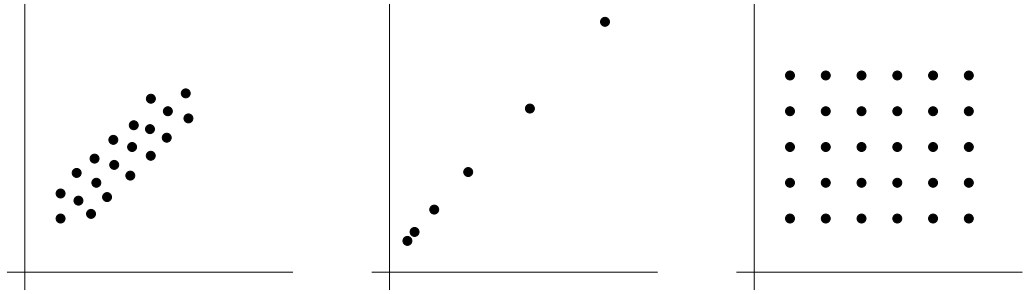
As the agglomerative clustering identification method is currently the only one of its type, and is without strong correlation to any other descriptor there seems little need for further discussion on its behaviour for the moment.

As might be expected, the descriptors within extent all correlate strongly with each other so choosing the best extent descriptor will simply be a matter of finding the one that can be computed in the fastest time. Of particular interest in the extent descriptors correlations is the fact that the estimation of the diameter correlates so strongly with the other extent measures – justifying its use as a measure of extent.

The dispersion methods are a 'mixed bag', encompassing a large range of descriptors which can provide information about the layout of a pattern, but do not necessarily fit in the more strictly defined classes. As a result it is not surprising that there is little correlation within this class. The two descriptors that show correlation are density and kurtosis. These also show strong correlation with the extent measures and, with a cursory examination, the fashion in which they are related is revealed. Excess kurtosis is a measure of how flat the distribution is (not strongly clustered around a single point), a pattern with a large extent is less likely to be strongly clustered[8] as there is more space in which the dots can exist. The density is clearly inversely proportional to the extent in that it is the cardinality divided by the area of the bounding box. Seeing that the kurtosis is linked so strongly to extent, the relationship between kurtosis and density is therefore made clear. The nearest neighbour descriptor shows a weak negative correlation with the dimensionality descriptors because the estimated nearest neighbour distance variance measures the uniformity of the patterns distribution, and the closer to collinearity a pattern is the less the variance of the distance between nearest neighbours is likely to be (Fig. 3.14(a)). However we can easily conceive of situations in which the nearest neighbour variance is high and the collinearity is high (Fig. 3.14(b)) or alternatively in which both are low (Fig. 3.14(c)). The exception case for high collinearity and high nearest neighbour variance is probably quite rare in real-world situations, however we suspect that the low nearest neighbour variance and low collinearity situation is not. Hence we do not remove the nearest neighbour variance in favour of a dimensionality measure or vice-versa. This does highlight a possible flaw in using heat maps in that they can show correlation that is only indicative of a trend in the test patterns, and as a result we tend to ignore any correlations that are only weakly negative or positive.

The dimensionality descriptors all show strong correlation as might be expected. It should

[8] Although it is possible to conceive of a pattern with a high proportion of the dots in the centre and a few outliers increasing the extent.



(a) Low nearest neighbour variance and high collinearity

(b) High nearest neighbour variance and high collinearity

(c) Low nearest neighbour variance and low collinearity

Figure 3.14 Examples of different values for estimated nearest neighbour variance and collinearity

be noted that we are using the absolute values for the coefficient correlations and returning a value of 1 when the pattern is completely horizontal or vertical. Without these amendments the coefficient methods can return negative values relating to the slope of the pattern and will return a NaN (Not a Number) value for linear horizontal or vertical patterns. In fact if these amendments are not made then the sample and Pearson descriptors both show correlation with the orientation methods. The eigenvalue difference measure, however, will show how 'linear' the pattern is regardless of its orientation.

The kind of analysis performed with the heat map shows the correlation between two descriptors, which is similar to the *influence* correlation considered by Andrienko and Andrienko [4]. However it does not provide information similar to the *structure* correlation in which two descriptors would have to interact and correlate with a third. For example, descriptor D_1 does not correlate with D_2 or D_3 but does with $D_2 \cup D_3$. This kind of correlation is not easy to identify but we examine how the selection of change identifiers, and thereby their descriptors, can be optimised to avoid this in Chapter 8.

With sets of descriptors that do not contain correlated measures the computation time for each set must be examined. Fig. 3.15 is a plot of the time taken in nano-seconds for each descriptor to compute a value for each phase in a randomised dynamic dot pattern set of length 3000 and a maximum pattern cardinality of ≈ 500 dots. For clarity the graphs have been plotted every 10 phases and then linearly interpolated.

There are three distinct 'bands' that are apparent in Fig. 3.15. Band 1 is the singleton containing just the agglomerative clustering descriptor, by far the most computationally expensive of the descriptors as it has a long iterative process. Even so the average completion time for the agglomerative clustering for each timestep is only around five seconds. Five seconds is probably too long a time for it to be used as a change identifier but for a pattern of approximately five hundred dots this is not an unreasonable time in which to find clusters given the computational complexity of the task.

The second band contains the descriptors for average nearest neighbour distance, the sample co-efficient correlation, the variance of the distance from the centroid and the skewness. All of the measures require more than one pass through the pattern: the first to find the mean, standard deviation, etc.; the second to use the values from the first pass.

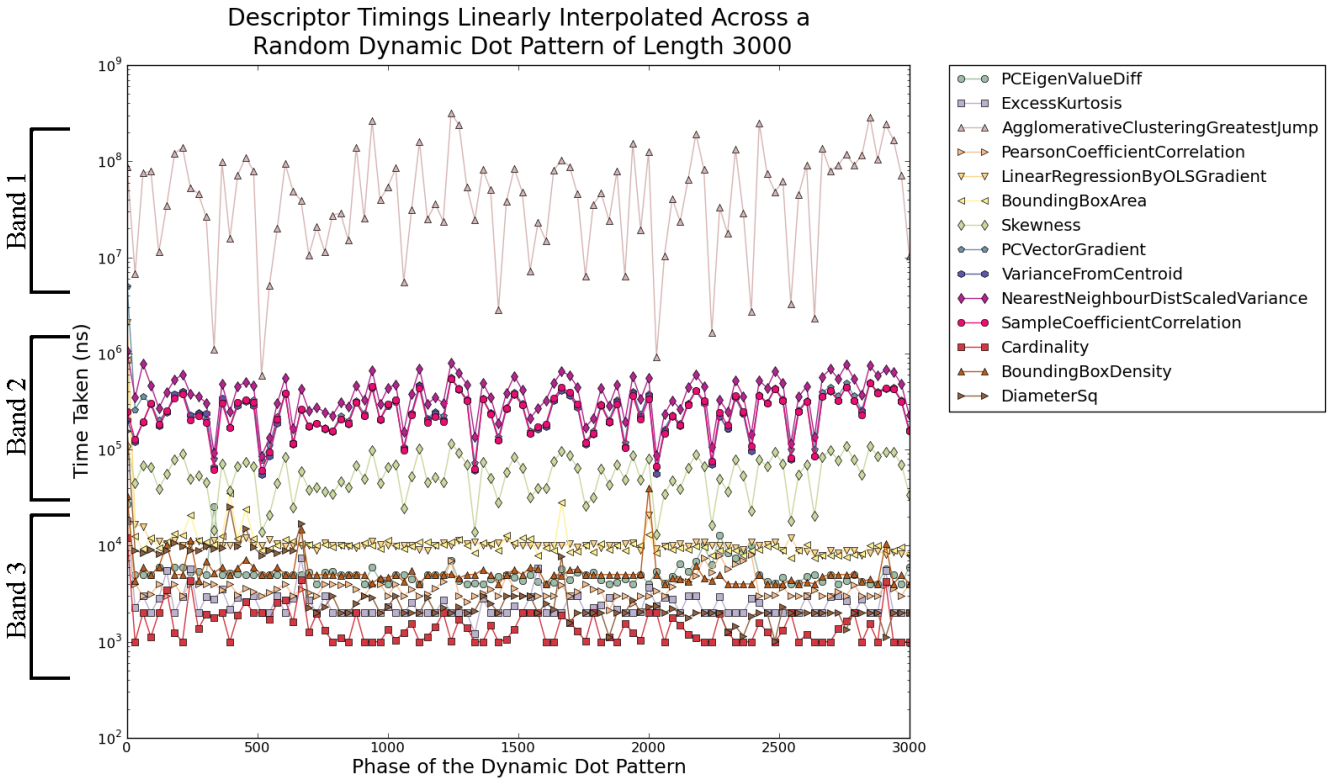


Figure 3.15 Time taken (nanoseconds) per dynamic dot pattern phase: All Descriptors

This may account for their taking longer than those in the third band despite many of the third having the same theoretical complexities.

The third band contains the rest of the descriptors. It should be noted that some of these descriptors share information and therefore we cannot say that any descriptor within a band necessarily outperforms another within that same band as their values may depend partially on the order in which they are processed.

The three-part banding indicates the fastest descriptors but the correlations show that some measure the same aspects of the pattern. The 'best' descriptor set to use will have the fastest descriptors with no correlations. The following graphs are plots of time taken against timestep for each of the descriptor classes so that we may find the fastest in each:

Extent: Fig. 3.16.

Orientation: Fig. 3.17.

Connectedness: No figure required as it is a class containing just the agglomerative clustering descriptor.

Dimensionality: Fig. 3.18.

Dispersion: Fig. 3.19.

As has already been discussed, dispersion shows little in the way of correlation apart from

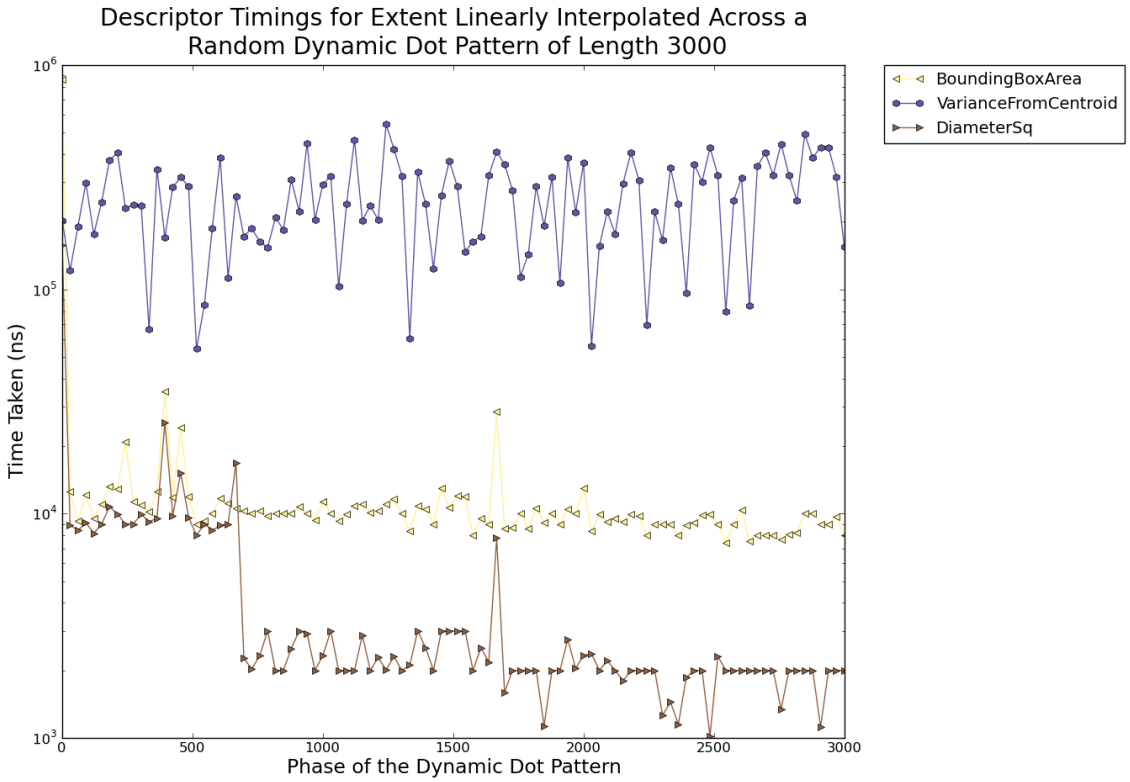


Figure 3.16 Time taken (nanoseconds) per dynamic dot pattern phase: Descriptors of Extent

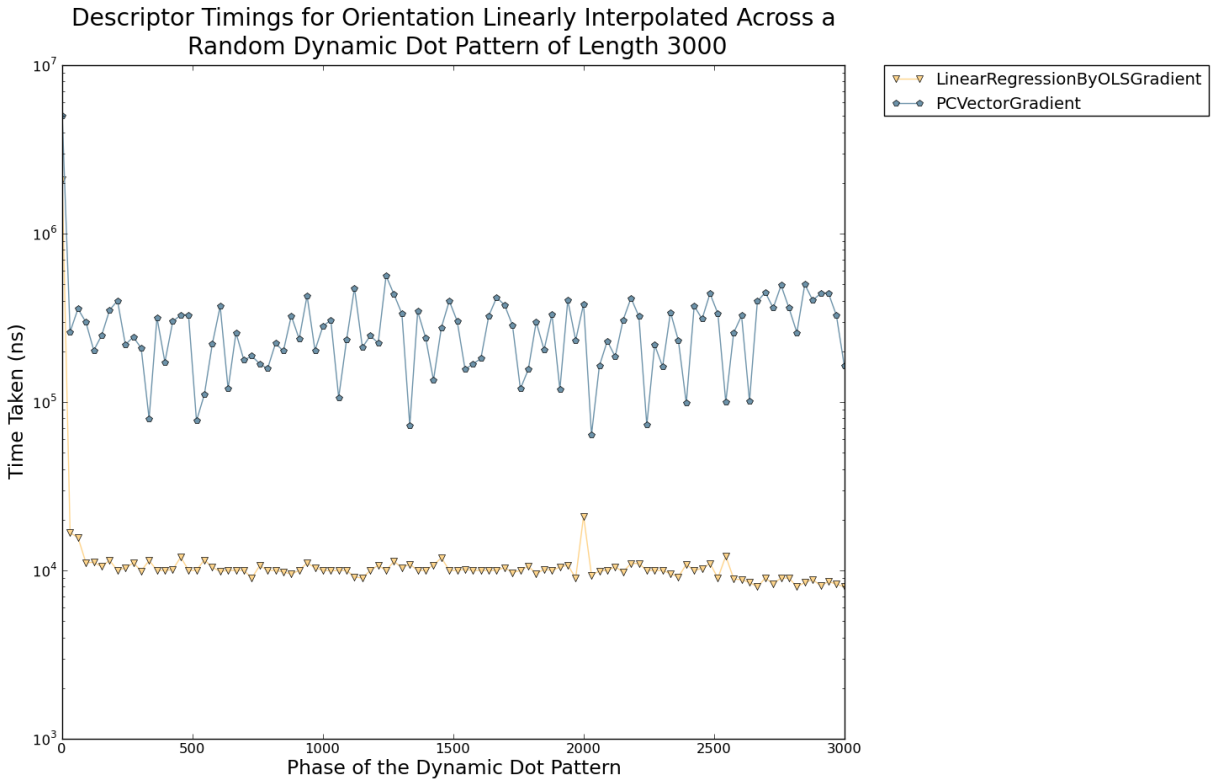


Figure 3.17 Time taken (nanoseconds) per dynamic dot pattern phase: Descriptors of Orientation

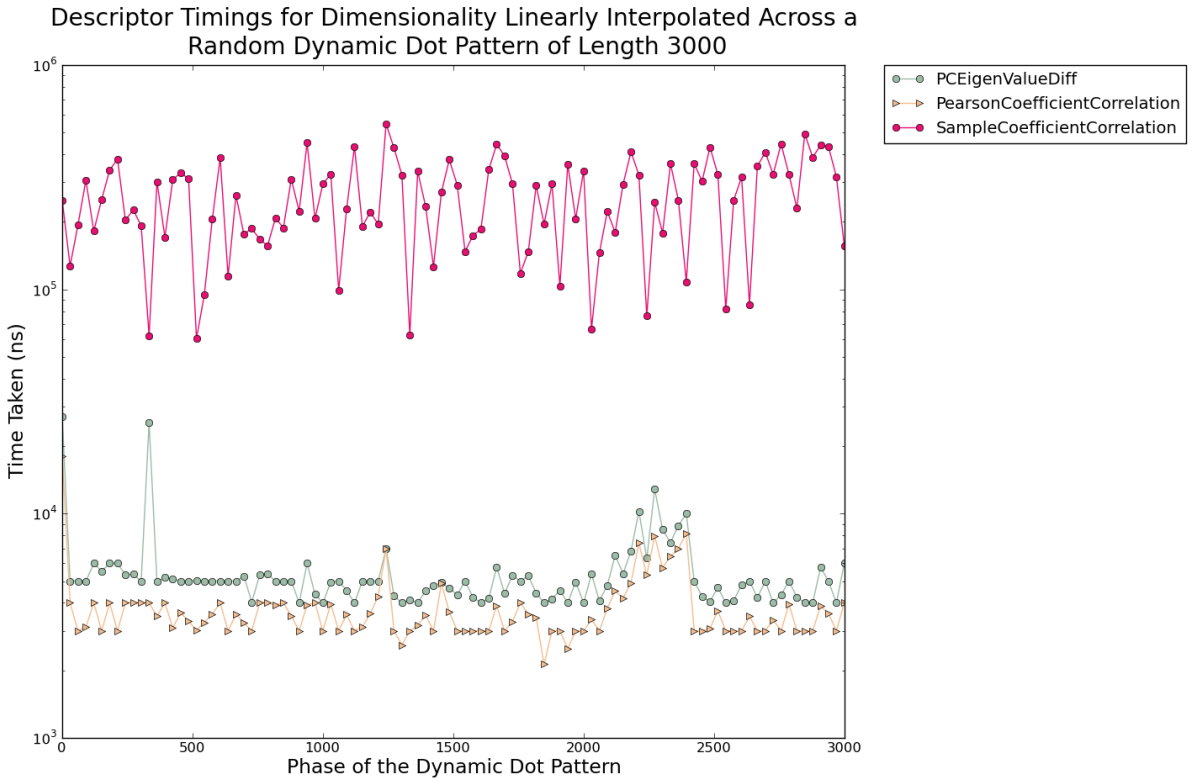


Figure 3.18 Time taken (nanoseconds) per dynamic dot pattern phase: Descriptors of Dimensionality

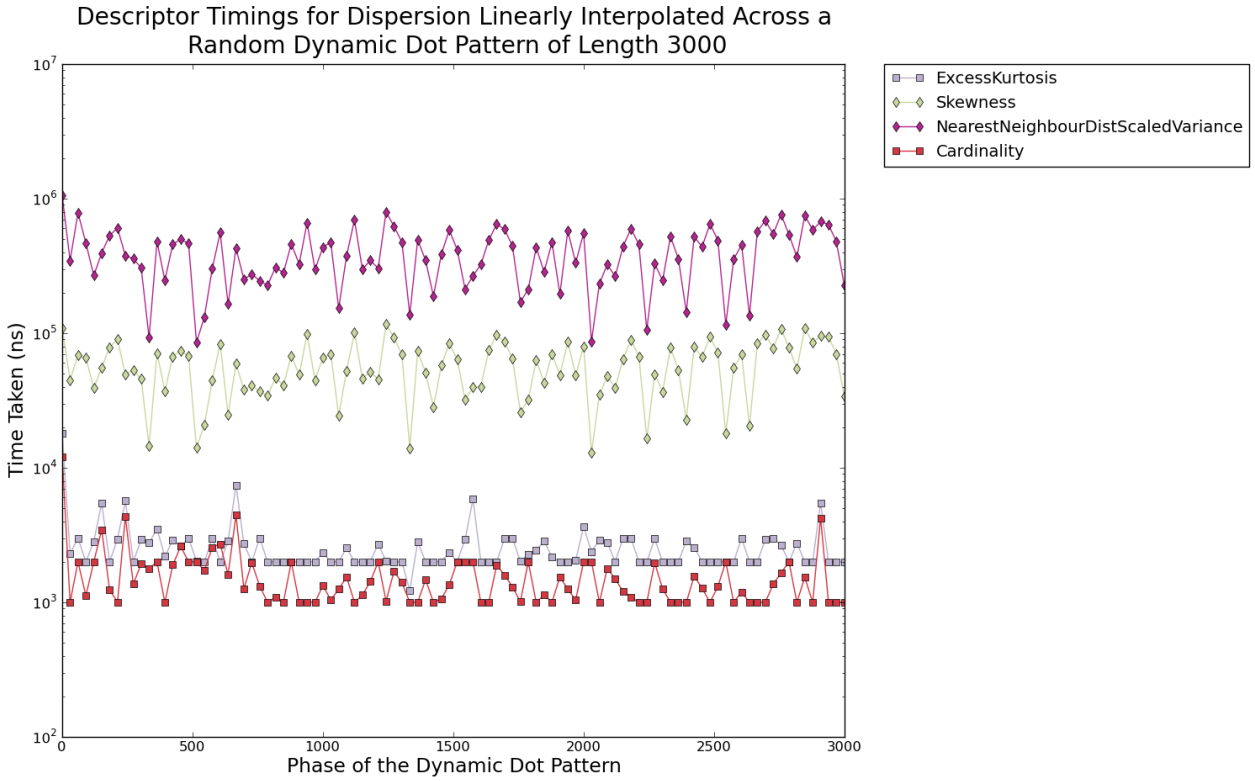


Figure 3.19 Time taken (nanoseconds) per dynamic dot pattern phase: Descriptors of Dispersion

the negative correlation of excess kurtosis with the bounding box density, but it has been included for completeness.

Fig. 3.20 shows the average time taken for each class against time steps. This is provided to give a sense of how similar the classes are in computation, apart from the class containing the agglomerative clustering descriptor.

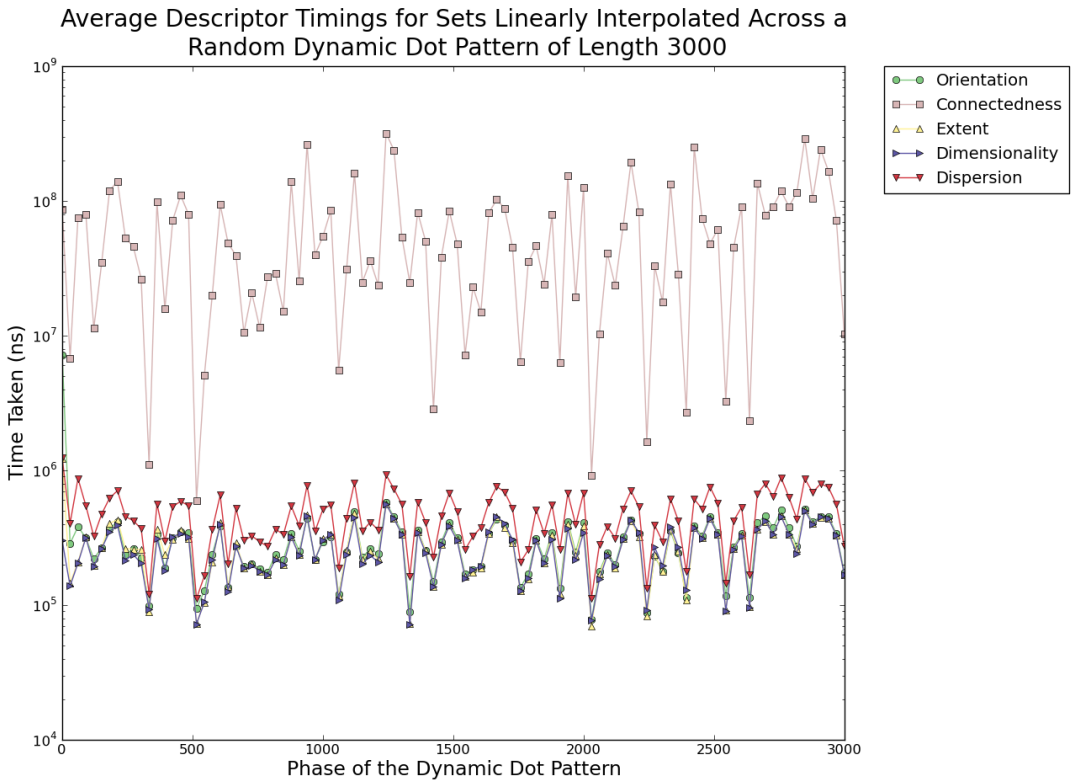


Figure 3.20 Average time taken (nanoseconds) dynamic dot pattern phase: Classes

With the information about the class speeds known we can create a set of the fastest completing descriptors from each class. This should give a good base set of descriptors on which we can construct change identifiers; completing in a minimum time while avoiding correlating, and therefore surplus, measures. Fig. 3.21 shows two sets using the fastest descriptors both with and without agglomerative clustering. For the change identifiers we will use only the set without the agglomerative clustering to avoid its high processing time.

3.5 Summary

This chapter has provided a list of descriptor classes and a set of non-correlated descriptors with a representative for each class. The descriptors have been timed and the fastest performing have been identified. It should be noted that speed is not the only indicator of quality for the descriptors. How fast they are to compute does not indicate the level at which they will be able to successfully identify change. To accurately measure this change

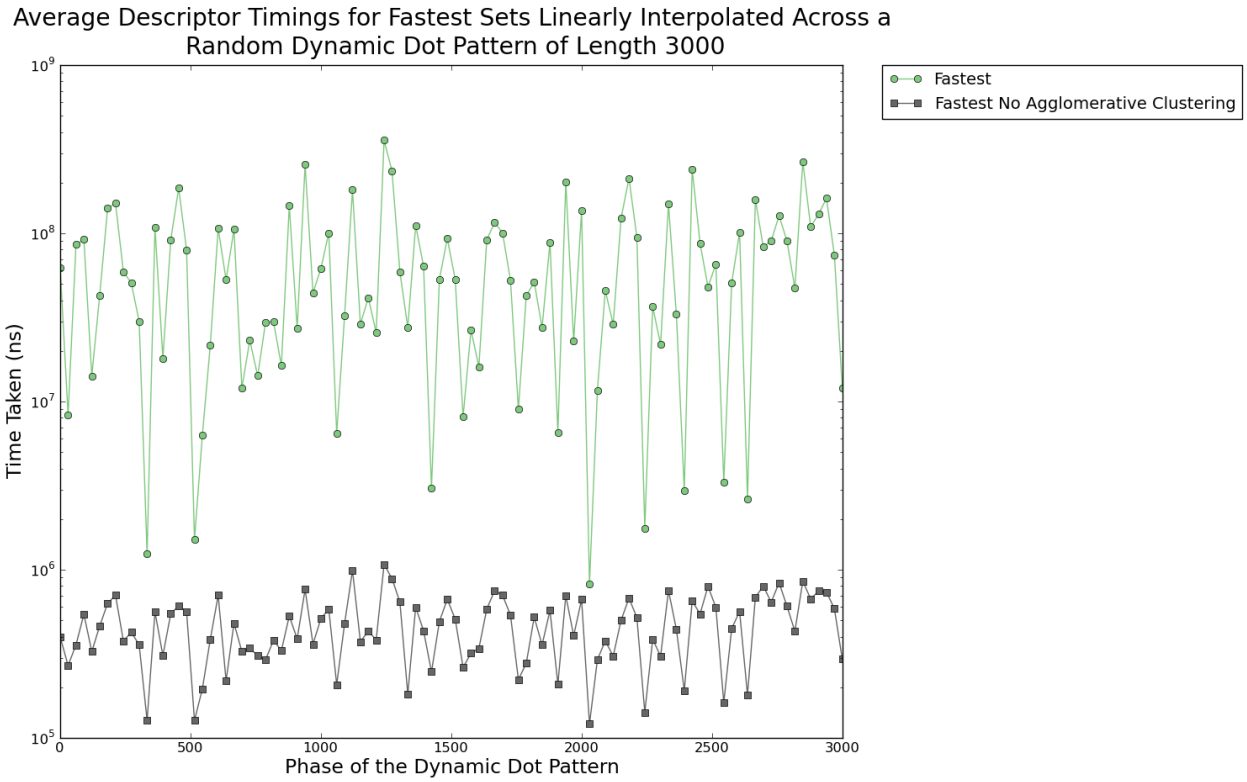


Figure 3.21 Time taken (nanoseconds) per dynamic dot pattern phase: Fastest non-correlated Descriptors

indication ability requires us to use the descriptors in the change identifier framework. We will examine this in Chapter 5 and show results of experimentation in Chapter 7. We content ourselves here with measuring the facets of the descriptors that exist regardless of change.

4 Footprints

The example use of change identifiers provided by this thesis is in the reduction of the number of updates of a footprint over a dynamic dot pattern. The previous chapter examined dot patterns and so, before discussing change identifiers, we must be sure we have a clear understanding of the nature of footprints.

The sheer number of algorithms described in the background chapter gives a good indication of the myriad applications in which footprints are required. These applications range from mapping molecular structure [24] to region approximation for geographic gazetteers [2]. Some of the papers make no mention of a specific application; inidicating that their algorithm is intended to be applicable to a range of different problems (e.g., Jarvis March [39] and χ-hull [20]). Given this 'broad strokes' approach taken across the field it is perhaps not surprising that there is a dearth of material that assesses why a footprint may be a 'good' footprint for any given application. This chapter provides a discussion on what a footprint is and a classification of the types of footprint that appear in the literature.

4.1 What is a footprint?

The existing literature has a strong tendency to use footprints that are both spatial and 2-dimensional, and it would be tendentious to start going too far beyond that which is required by the current applications. We specify that, for this thesis at least, the footprint be spatial and cannot have a dimensionality that exceeds that of the space in which the dot pattern upon which it is formed is embedded (for example a 2-dimensional pattern can have a 1 and a 2-dimensional footprint but not a 3-dimensional one). Further we allow for degenerate lines but ignore internal partitions of a footprint. For example the Delaunay triangulation should not be considered as a footprint, but the union of the closed triangles of the Delaunay triangulation would be.

There are some footprint types for a pattern which are distinctive. Any footprint with a specification that is unique on a particular dot pattern is, at least intuitively, different to a general footprint. Perhaps this is easier understood by the observation that there are some footprints which are named and the name uniquely defines them for a pattern. For example the convex hull, the minimum bounding disc and the minimum bounding rectangle. We add the *identity* footprint to these as the footprint for which each dot is a distinct component of the footprint with no extent; in effect the dot pattern itself. It may be that such footprints are only unique in that they fit with some human understanding, for example why are they any different from a footprint named "The α-hull for an α

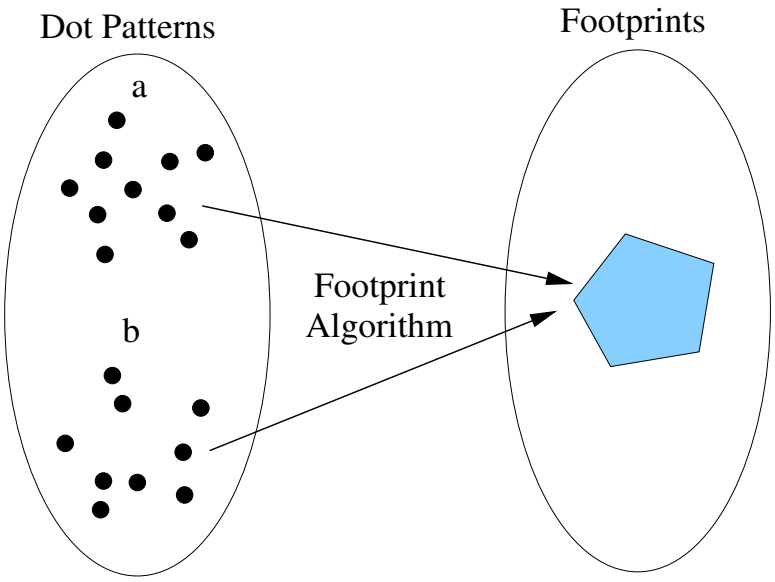


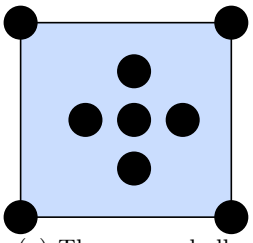
Figure 4.1 Figure showing that a footprint algorithm is not injective, both dot patterns a and b map to the same footprint.

value of 0.1"? However such a description specifies the value of the property used by an algorithm whereas the convex hull, minimum bounding disc and minimum bounding rectangle give no such properties, nor do they dictate the algorithm to be used. Hence, the key difference would appear to be the distinction between specifying a footprint by its properties versus specifying a footprint by an algorithm. The unique footprints we give as examples are all minimally bounding with respect to the dots, this is not necessarily true for all unique footprints. For example, the circle of maximum radius that sits within the convex hull of the dot pattern, although we have not come across any examples of unique footprints in the literature that are not minimally bounding.

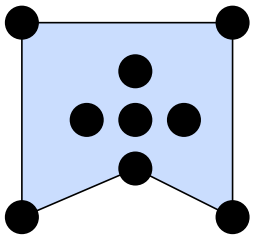
As a representation of the pattern the information content of the footprint is a measure of how well it defines that pattern. Note that this says nothing about the quality of the footprint which, as discussed, is difficult to judge, but is more an indication of how well the dot pattern could be re-created given just the footprint. We draw a distinction between the information content of the footprint and that of the dot pattern; while the footprint is created from the pattern, the mappings implied by the footprint algorithms are not injective (Fig. 4.1). There is a loss of information in that, even given the algorithm (and parameter if required) which created it, no footprint uniquely defines a pattern (except the identity footprint). However, given a specific algorithm with determined parameters a pattern uniquely defines one footprint. Importantly it should noted that the loss of information is not the same as having less information. The number of bits of data required to draw the footprint is not representative of the ability to retrieve the dot pattern from the footprint and can be greater than that required to draw the dot pattern. Yet, for most sensible examples[1], how well the dot pattern information can be estimated from a footprint is often proportional to the number of bits required to draw it. For example a sample of points within the footprint of Fig. 4.2(c) will be closer to the actual dot pattern than a

[1]It is easy to add extra information to a footprint without increasing the ability to retrieve the dot pattern by adding lines that render the footprint a worse representation

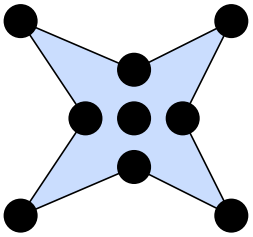
sample of points within the footprint of Fig. 4.2(a). While we have been careful to show examples where a footprint specification requires more information than a dot pattern, for dot patterns with areas of high density the data content of the footprint is unlikely to be greater than that of the dots; as the density increases so does the likelihood that fewer dots are on the boundary of the footprint. Some footprint algorithms (e.g., DSAM [2]) are created explicitly to produce a representation with a lower memory requirment than the dot pattern. Therefore for most real-world applications in which footprints may need to be found it is likely that footprints with less data content than that of the dot pattern is preferred. Fig. 4.2 shows the differences in information content that different footprints on the same pattern can have.



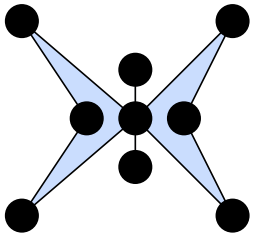
(a) The convex hull.



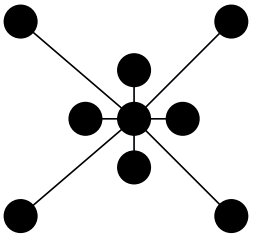
(b) The footprint has a greater information content than the convex hull and more closely represents the pattern.



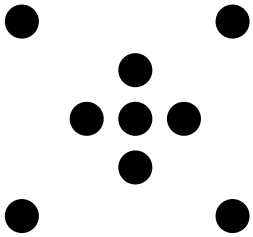
(c) Depending on how the footprint is stored, this footprint can use more memory than the dot pattern.



(d) The footprint is still a single component but it is no longer a jordan curve.



(e) Dots are joined by degenerate lines; the footprint has no area.



(f) Each dot is a component. The information content of the footprint and the dot pattern are identical as the footprint is the identity footprint.

Figure 4.2 Changes in information content for a footprint over a small dot pattern

The complexity of a footprint algorithm is closely tied to the information in the footprint it produces; as the amount of information in the footprint increases there will be a tendency for the computational complexity of the algorithm to increase. Largely this is due to the need for iterative processes and validity checks. For example the χ-hull is more computationally complex than the Jarvis March/Gift-Wrapping algorithm which is, in turn, more computationally complex than an algorithm for defining the isothethic minimum bounding box. This is not a true relation but an inclination of the algorithms in the literature; some algorithms have low information content but a high complexity, for example an algorithm

to find the minimum bounding disc. An algorithm to reproduce a dot pattern need only record the location for each dot, and as such, footprint algorithms are generally far more complex. While it is possible to come up with footprint algorithms less computationally complex than a dot pattern recreation algorithm, this tends to involve having a footprint algorithm ignore the dot pattern (e.g. it simply draws a square with a set length with its top left corner being the first dot it encounters), or by adding wasteful steps to the dot pattern algorithm. As a final point on information and how it applies to footprints we note that an algorithm to define the identity footprint (the trivial footprint in which each dot is its own component) produces a footprint with identical information to that of the dot pattern (Fig. 4.2(f)) with a computational complexity of $O(n)$; it simply needs to record each dot location. Such an algorithm is bijective (as a dot pattern has only one identity footprint and an identity footprint has only one dot pattern) and therefore unusual amongst the footprint algorithms found in the literature.

4.2 Footprint Classification

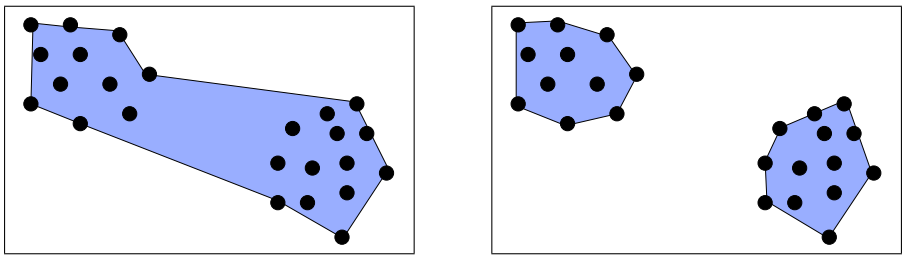
In the background chapter a summary was given of work performed by Galton and Duckham [28] and Galton [27]. These papers provide an insight into some of the aspects that need to be addressed when assessing the quality of an algorithm. Building on the work by Galton and Duckham, this author and Galton produced a classification [21] by which to draw distinctions between different footprint types. The footprints are delineated into classes with the aim of being able to provide reasons why they might, or might not, be suited to a specific application. This section of this chapter will discuss and extend the footprint classification.

The classifications have been split into two sections: *intrinsic*, concerning the footprint independent of the dots, and *relational*, examining the relationship between the dots and the footprint. When the footprint is expected to change, the delineation between intrinsic and relational classifiers becomes more distinct. A change in any of the intrinsic values would indicate a complete change of footprint type. Any relational change is minor and to be expected when using change identifiers, for example whether or not all dots are within the boundary of the footprint is almost certain to change when using a dynamic dot pattern. It should be noted that the classification is not an exhaustive taxonomy of shape but showcases the features drawn from the common differences between the footprints created by the algorithms in the existing literature.

4.2.1 Intrinsic footprint criteria

[C] Connected: *The footprint consists of a single connected component.*
Figure 4.3 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 [50], χ-shape [20]); others can yield footprints

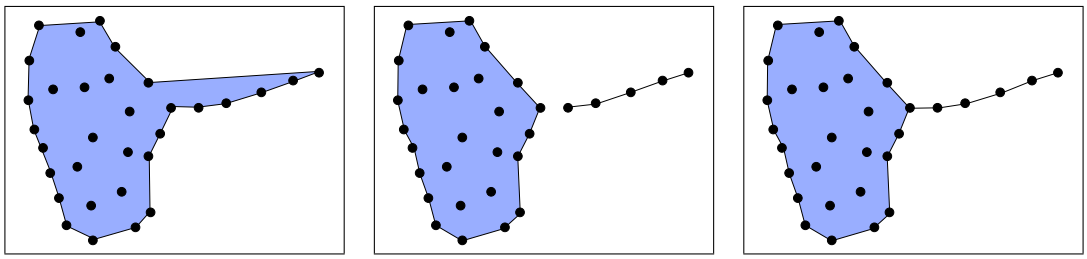
with multiple components (e.g., Swinging Arm [28]). The desirability or otherwise of multiple components is application-dependent, e.g., if only connected footprints are appropriate, use an algorithm guaranteed to produce such components.



(a) Connected [C] (b) Multiple Components [¬C]

Figure 4.3 Connectedness examples

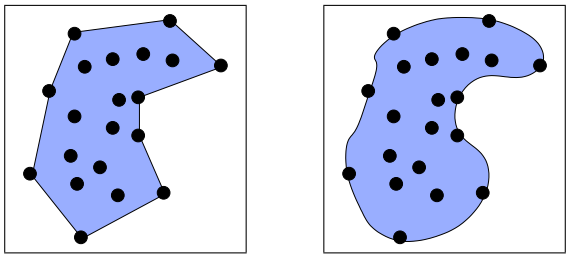
[R] Regular: *The footprint is topologically regular.*
Assuming the footprint is topologically closed, this criterion tells us whether or not the footprint contains boundary elements that do not bound the footprint's interior, such as the linear 'spike' in Fig. 4.4(c) or the isolated linear component in Fig. 4.4(b).



(a) Regular (b) Irregular and disconnected [¬R ∧ ¬C] (c) Irregular [¬R]

Figure 4.4 Regular

[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. (Fig. 4.5).

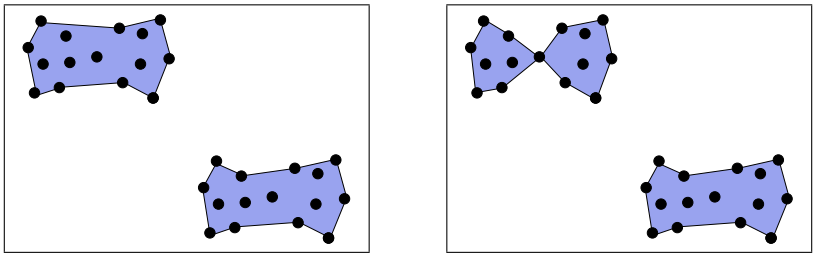


(a) Polygonal [P] (b) Curvilinear [¬P]

Figure 4.5 Polygonal

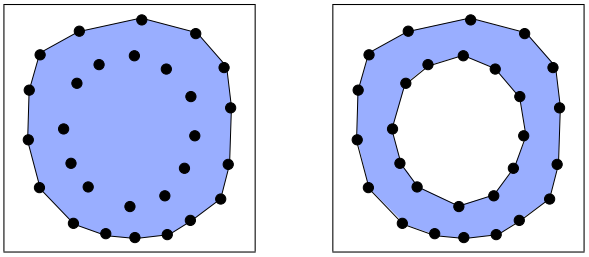
[JC] Jordan Components: *Each component of the footprint has a Jordan boundary.*
A Jordan boundary is a boundary which is a Jordan curve, i.e., homeomorphic to a circle. Such a boundary does not meet itself, so it is possible to traverse the entire boundary passing through each of its points only once. (Fig. 4.6(a)). In Fig. 4.6(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.[2]



(a) All Jordan Components [JC] (b) Not all Jordan Components [¬JC]

Figure 4.6 Jordan Boundary



(a) Simply Connected [SCC] (b) With Cavity [¬SCC]

Figure 4.7 Simply Connected

[SCC] Simply Connected Components: *Each component of the footprint is simply connected.*

A component that is not simply connected contains a 'hole' (Fig. 4.7(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.[3]

4.2.2 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 (Fig. 4.8(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

[2] 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.

[3] In three dimensions there are more varieties of connectivity to consider, e.g., the distinction between an internal cavity and a perforation. For simplicity (and because the majority of algorithms are in 2-dimensions) we do not discuss these extensions here.

by Fig. 4.8, where each value for one criterion can co-occur with each value of the other. However, [CED] ∧ [NDB] (all curvature extrema are dots and all dots are off the boundary) can only be true if the footprint is circular, in which case there are no curvature extrema, so [CED] is true by default.

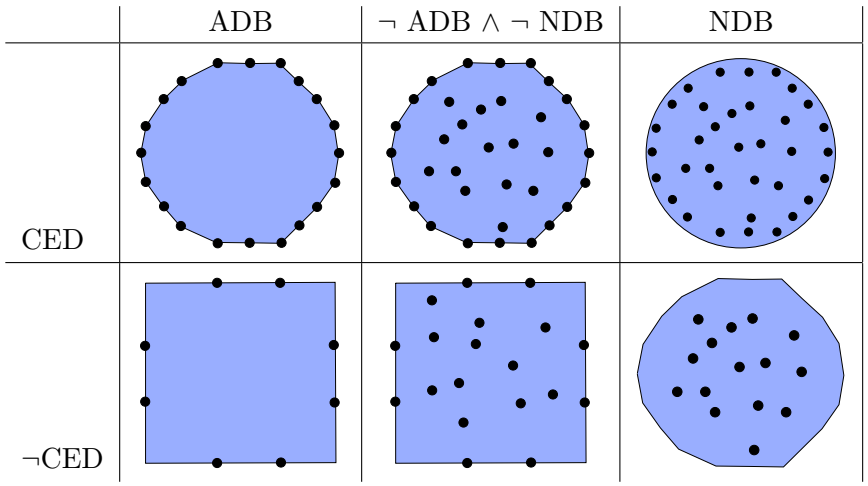
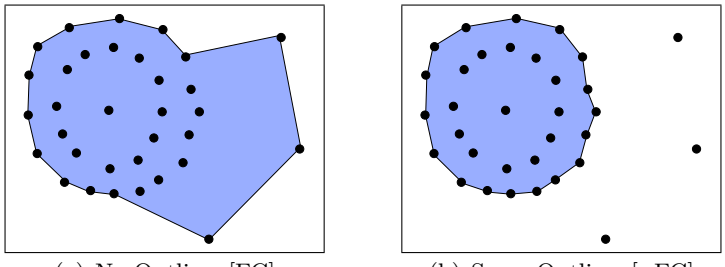


Figure 4.8 Curvature Extrema and Dots On/Off Boundary

[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 (Fig. 4.8).

[NDB] No Dots on Boundary: *None of the dots lie on the boundary of the footprint.* Criteria [ADB] and [NDB] cannot be simultaneously satisfied, thus they are not independent. As with [ADB], [NDB] is linked to, but distinct from, whether or not the curvature extrema coincide with dots (Fig. 4.8) as both [NDB] and [CED] can only be true for a circular footprint. Some algorithms (e.g., the Voronoi-based method of [2]) 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.



(a) No Outliers [FC] (b) Some Outliers [¬FC]

Figure 4.9 Full Coverage

[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* (Fig. 4.9).

4.3 Using the Footprint Classification

With a set of criteria by which to classify the footprints a nomenclature can be created to easily distinguish between individual footprints. When classifying a footprint the values can be written in the form presented below:

T_x: Footprint Type (intrinsic) x

F_τ: Footprint at time τ

$I(v, x)$: Value of intrinsic classifier v for type x

$R(v, \tau)$: Value of relational classifier v at time τ

As discussed earlier, the intrinsic classifiers are the more 'concrete' classifiers and the relational are more prone to change with time. This distinction can be used to produce a listing to describe the footprint at a given time:

$$T_1 = \{I(C,1), I(R,1), I(P,1), I(JC,1), I(SCC,1)\}$$
$$F_\tau = \{T_1, R(CED,1), R(ADB,0), R(NDB,0), R(FC,1)\}$$

A shorthand representation can be created by requiring that the intrinsic and relational classifiers always appear in the given order {C,R,P,JC,SCC} and {CED,ADB,NDB,FC} respectively:

$$T_1 = \{1,1,1,1,1\}$$
$$F_{\tau_x} = \{T_1,1,0,0,1\}$$

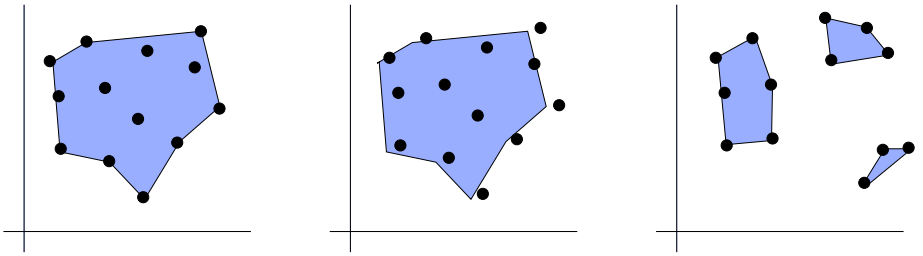
The nomenclature allows the tracking of the footprint type over a dynamic dot pattern and an example of this is shown in Fig. 4.10. If $T_1 = \{1,1,1,1,1\}$ then at dot pattern phase ϕ_0 (Fig. 4.10(a)) the footprint is of type $\{T_1,1,0,0,1\}$. The full tracking of the footprint classifications of Fig. 4.10 are shown in Table 4.1.

Intrinsic Type	Classification
T_1	$\{1,1,1,1,1\}$
T_2	$\{0,1,1,1,1\}$

Fig. Ref.	Phase	Classification
Fig. 4.10(a)	ϕ_0	$\{T_1,1,0,0,1\}$
Fig. 4.10(b)	ϕ_1	$\{T_1,0,0,0,0\}$
Fig. 4.10(c)	ϕ_2	$\{T_2,1,0,0,1\}$

Table 4.1 Table showing classification of Fig. 4.10

Before further examining the application of the classification to dynamic dot patterns we look at what is perhaps a more generally useful aspect of the classification. As previously mentioned, there are a large number of algorithms for multiple applications. The algo-



(a) Dot pattern phase ϕ_0 (b) Dot pattern phase ϕ_1 (c) Dot pattern phase ϕ_2

Figure 4.10 Footprint type tracking example.

rithms, by virtue of their construction, are limited in the footprints they can produce for any given dot pattern[4]. We can, at least in part, classify the footprint algorithms by the footprints they produce[5]. Instead of the true/false value, used by the footprint classification, the algorithms require a ternary system where -1, 0 and 1 indicate that classifier never holds true, sometimes holds true and always holds true respectively. We introduce the shorthand of [algorithm] is [property] to indicate that the footprints produced by that algorithm have that property. For example the Swinging Arm algorithm ([28]) is always regular ($I(R, 1)$ in the above nomenclature). More accurately the Swinging Arm algorithm is regular except when the dots of the pattern are collinear; the correct value, then, should be 0. Other algorithms have similar changes in value for particular pattern arrangements and the value system becomes worthless if all the values are 0 so we add two special case values: -1^+ if the classifier is never true for 'almost all' patterns, where 'almost all' excludes specific and unlikely to occur pattern arrangements (e.g., collinearity, the null pattern[6] and a pattern with only one dot); and 1^- for when the classifier is always true for 'almost all' patterns. Some examples of this nomenclature can be found in Table 4.2[7].

Algorithm Examples	C	R	P	JC	SCC	CED	ADB	NDB	FC
Jarvis March [39]	1	1^-	1	1^-	1	1	0	-1	1
Swinging Arm Algorithm [28]	0	1^-	1	1^-	1	1	0	-1	1
α-shape [23]	0	1^-	1	1^-	0	1	0	-1	0
χ-hull [20]	1	1^-	1	1^-	1	1	0	-1	1
DSAM [2]	1	1	1	0	0	-1	-1	1	1

Table 4.2 Algorithm Classification Examples

In classifying the footprints we have given an indication of the range of possible footprint types that can be created. The type of footprint on a dynamic dot pattern can be tracked using the nomenclature provided. While many of the relational criteria would be expected to change (for example, in Fig. 4.10(b) the dots have moved from inside to outside the footprint), if an intrinsic classifier value changes it is an indication that a large change has occurred in the dynamic dot pattern (for example, in Fig. 4.10(c) the footprint is

[4]Some algorithms can produce an infinite number of footprints for a given dot pattern (e.g., α-shapes [23]), however they cannot necessarily produce all possible footprints.
[5]A full classification would describe their complexities, pre-processing requirements and perhaps some description of their running process
[6]An empty pattern
[7]In which the 1^- on regularity and Jordan components refers to patterns with collinear dots.

no longer a single connected component). As discussed in Chapter 2, many footprint algorithms require an external parameter (e.g., α in α-shapes [24], line length in the swinging arm algorithm [28], etc.). Most of the papers describing these algorithms come to the conclusion that for their algorithm there is no generally systematic method that can choose an appropriate parameter. Those that do provide such a method involve an iterative construction of the footprint changing the parameter each time until it satisfies some constraints (for example Chaudhuri *et al.*'s s and r-shapes [14]). In a dynamic dot pattern the changes in the pattern will likely mean that the parameter, even if suitable at the first time step, will need to be changed and this is almost certain if the change has been great enough that the footprint has changed in intrinsic type. We will look at the cost and potential benefit of identifying the change in intrinsic footprint type further in the future work chapter.

4.4 Summary

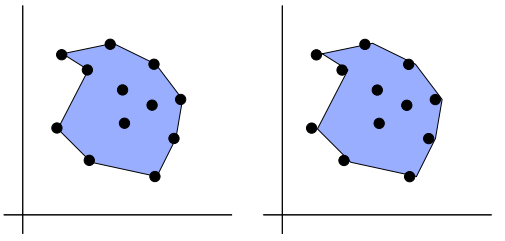
This chapter has provided a detailed look at footprints and how they can be classified. It has discussed the possible uses that such a classification may have and how it can be applied to dynamic dot patterns. Importantly the investigation of footprints, in conjunction with the examination of dot patterns, gives the forthcoming examination of change identifiers a solid foundation. This chapter and Chapter 3 have discussed the outputs and inputs respectively of the framework within which the change identifiers will be expected to operate.

5 Change Identifiers

The thesis has now covered the requisite background information for the change identifiers to be formally introduced and examined. This chapter will present, in greater detail than previously given, the reasoning behind and within the construction of the change identifiers.

5.1 What is a Change Identifier?

Changes in collectives will correspond to changes in the geometric (and statistical) properties of its representative dynamic dot pattern. As discussed earlier it is possible that the change in a dynamic dot pattern between two timesteps is small or even non-existent, for example the phases shown in Fig. 5.1. For such small changes re-computing the footprint at each timestep is unnecessary computation. In the background chapter it was observed that there is current research into the use of spatio-temporal data techniques in emergency situations (wild fires, chemical spills, etc. [40, 41]). These emergency situations require a fast and appropriate response, but even when used in non-emergency based applications (such as herd tracking) a fast and appropriate response is desired. If the footprint algorithm takes longer to run than the speed at which the dot patterns arrive then the representation falls behind the actual state of the phenomenon. Table 5.1 shows an example of this when the footprint takes twice as long to run as the time taken for a dot pattern to arrive. The first column is the current time step, the second is the time step that the current footprint is a representation of and the third column shows how far behind the current state the footprint is.



(a) Initial positions for phase ϕ_0

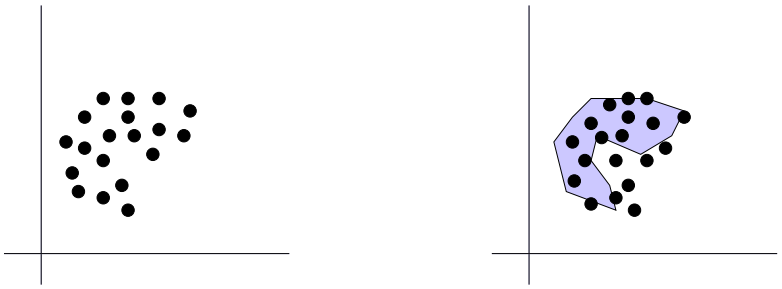
(b) Dot pattern has moved slightly for phase ϕ_1

Figure 5.1 Has the dot pattern at ϕ_1 changed sufficiently for the footprint to be updated?

The figure Fig. 5.2 shows how this lag would appear following the same pattern as that given in Table 5.1.

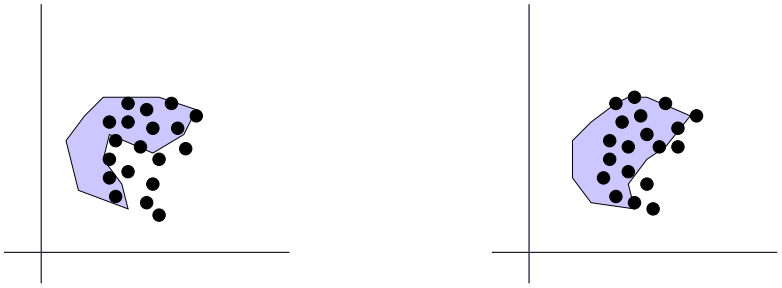
Time Step	Representation	Lag
1	-	-
2	1	1
3	1	2
4	2	2
5	2	3
6	3	3

Table 5.1 The patterns arriving at twice the speed it takes a footprint to be computed.



(a) Time step 1: The footprint has not yet been created

(b) Time step 2: The footprint has been created and is representative of the dot pattern at time step 1



(c) Time step 3: The footprint is still representative of the dot pattern at time step 1

(d) Time step 4: The footprint is now representative of the dot pattern at time step 2

Figure 5.2 Figure showing increasing lag in footprint representation

In any application, when the representation falls behind in this fashion it hinders the user's ability to make decisions about the data. By reducing the number of required footprint updates the change identifiers allow the system to produce accurate footprints faster. For the user, this means that at any given timestep the displayed footprint can be trusted as an acceptable representation.

Often the requirements of an application are not limited to just visualising the region of interest but to use this visualisation to make decisions about how the events underlying the dynamic dot pattern are changing. For example if tracking the spread of a crowd in a shopping centre the user may want to know when it is forming clusters. The change identifiers can provide this information without the user having to interpret it from the footprint. We will discuss this further when we consider possible future work (Chapter 10).

Once the method by which to calculate the identifiers has been decided upon, we must provide a cognitively salient way for a threshold to be set. Salience and its importance has previously arisen within Chapter 2 when footprint algorithms and their parameters were

discussed; the user has to set a parameter for both change identifiers and the footprint algorithms. The difficulty that arises comes from the generality of the expected uses for footprint algorithms and change identifiers; different applications will have different requirements. Parameters are necessary to allow for these different requirements, however they add a layer of complication for a user. For example given a dot pattern of 1000 dots, ascertaining the correct value of α to use to produce the ideal α-hull for an application trying to find the boundary of a city is not an easy task. In practice such parameters tend to be decided on by trial and error. With change identifiers there is the understanding that time is a constraint, and therefore such trial and error approaches are probably infeasible. By making the threshold setting as cognitively salient as possible the difficulty in choosing an appropriate threshold can be greatly reduced. Ideally the change identifiers should be able to have a threshold set in a way that is intuitive no matter what application they are being used in.

Given a change identifier that measures the area difference of the bounding box of the pattern we can demonstrate the problem in using the area value as the measurement returned by the identifier.

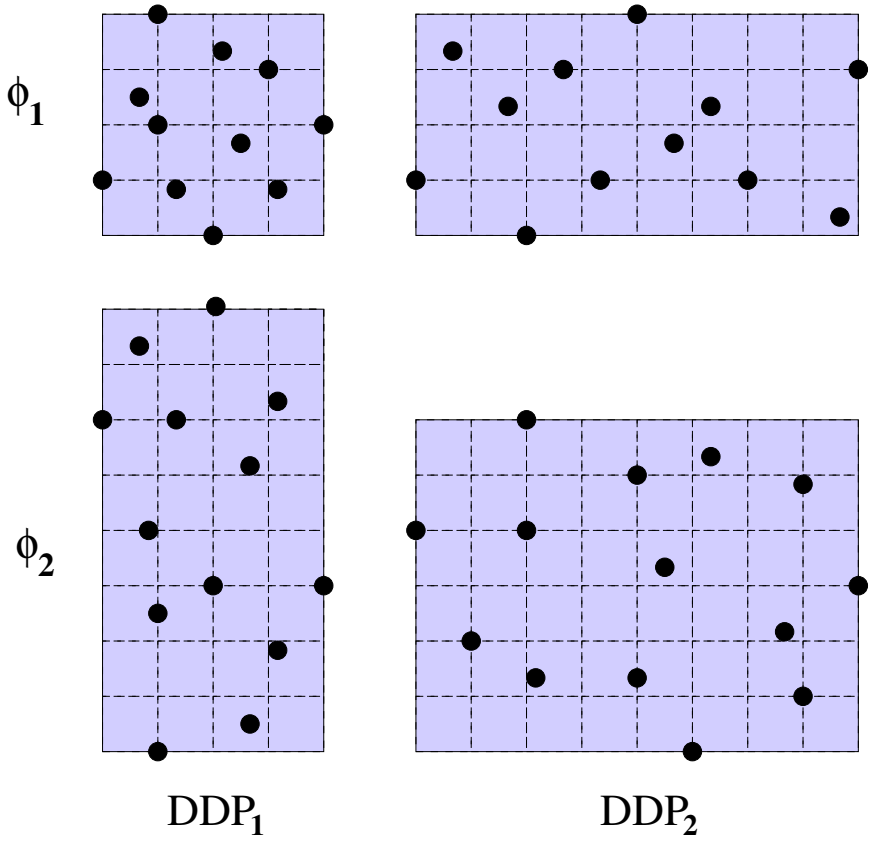


Figure 5.3 Difference in bounding box area for two dynamic dot patterns.

Fig. 5.3 shows two phases (ϕ_1 and ϕ_2) from two dynamic dot patterns (DDP_1 and DDP_2). The area change from ϕ_1 to ϕ_2 is the same for both (an increase of 16 units^2), however DDP_1 has doubled in size whereas DDP_2 has only increased by 50% of its original size. It would not be unfair to state that the change in DDP_1 has a greater impact than the change in DDP_2. A threshold is required to signal the framework that the footprint must be updated when sufficient change has occurred. Should the threshold be 'concrete' (i.e.

it is a fixed value and not relative to the dot pattern) then the identifier is not tracking the impact that the change is having on the dynamic dot pattern. For example, if using the bounding box area, crossing a concrete threshold will represent different impact levels of change because the size of the dot pattern at each phase changes.

The thresholding concern can be satisfied if the requirement that all identifiers return a proportional value is introduced; a value that represents proportional change in the property the identifier measures since the last timestep at which the footprint was updated. The threshold now becomes a percentage value and is, therefore, more cognitively identifiable than a concrete value. For example, in the bounding box example (Fig. 5.3) the user could specify that the footprint is updated when the bounding box area has changed by 100%; triggering an update in DDP_1 but not in DDP_2. While this still relies on user input we feel that it represents less of a mental leap than intuitively 'knowing' by how many units^2 a pattern's bounding box area will need to change before its footprint is no longer a suitable representation. The definition of an identifier can now be formalised as:

Definition:

> A change identifier is a measurement that compares two phases of a dynamic dot pattern (ϕ_1 and ϕ_2) and returns the difference in a descriptor of the dot patterns expressed as a proportion of the value of that descriptor on the dot pattern ϕ_1.

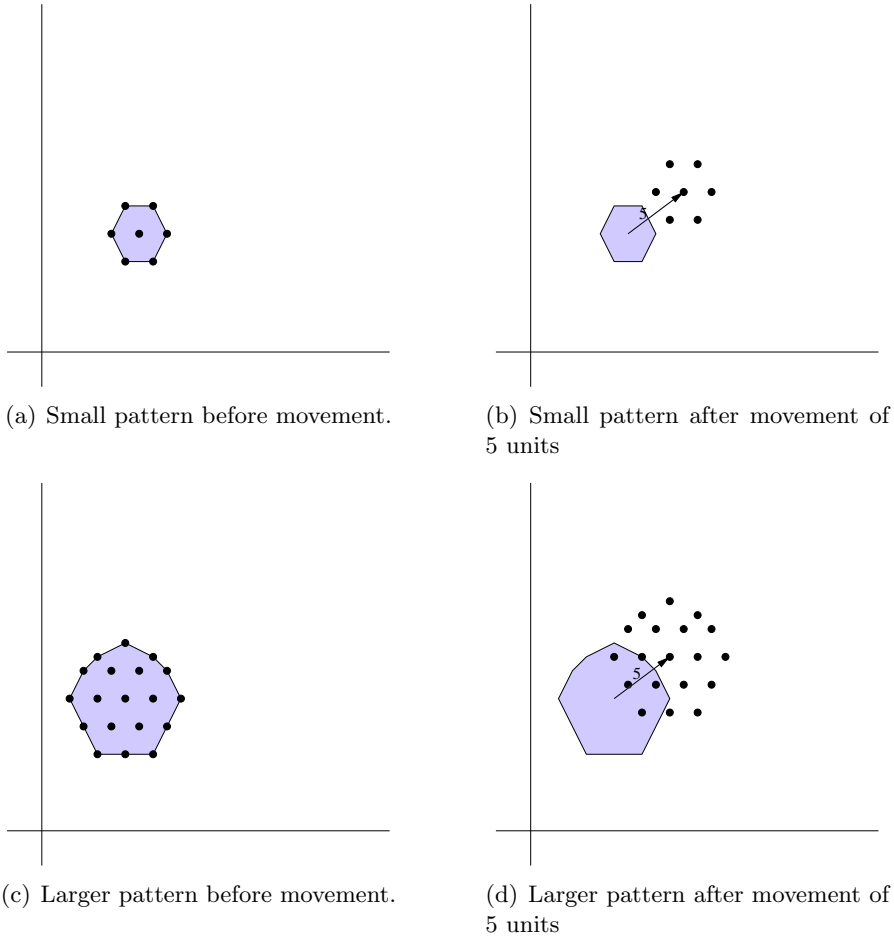
For the purposes of this thesis we can add the requirement that the change identifier must be computable in less time than it takes to recompute the footprint. In practice the identifier should ideally have a complexity such that its value can be found in less time than a footprint algorithm with complexity $O(n \log h)$[1] takes to produce a footprint, this being the optimal time complexity yet found for a convex hull algorithm. The convex hull time is used as the maximal allowed time taken for computation for an identifier as it is fast compared to the majority of footprint algoritms but not so fast that it is infeasible to compute a change identifier measurement in less time (as opposed to, for example, an isothetic bounding box algorithm).

5.2 Metrics

When considering potential change identifiers one of the most immediately obvious candidates that will need to be measured is change in location. While location maybe an intuitive starting point it has two unique aspects, primarily because we can use it to update the footprint without recomputation; translating the footprint along the same vector that the dynamic dot pattern has moved. The other standard transformations (scaling, rotation and shearing) can also be directly applied to the footprint in a computationally fast time, however identifying change in these transformations is less easy and none of

[1] Here n is the number of dots and h is the number of dots at the vertices of the footprint.

them are as simple to apply to the footprint as the change in location. The other interesting aspect of a location change identifier is that it shows a complication in the use of the difference in descriptor values as the change measure. Location is a measurable property of a dot pattern but, instead of a single real-number, it is a vector of dimension equal to the space the pattern resides in. The vector adds complexity as scaling it and checking it against a threshold can not be done in the same fashion that is provided in the original change identifier definition for two reasons. Firstly it would require the use of a vector as a threshold, as mentioned above we wish to make the choice of thresholds as simple as possible and having thresholds of differing types runs counter to this aim. Secondly, and more importantly, it makes no sense to make the change in position proportional to the value of the position at the earlier phase as this has no bearing on how much the dynamic pattern has changed between the phases. This can be generalised with the statement that any change identifier should be invariant under a change of coordinate system. Taking the



(a) Small pattern before movement.

(b) Small pattern after movement of 5 units

(c) Larger pattern before movement.

(d) Larger pattern after movement of 5 units

Figure 5.4 Figure showing change in effect for a change in position for two different sized patterns

distance between two locations would be a sensible way to reduce it two a 1-dimensional value but doing so loses information about the direction in which the dynamic pattern has moved. A further consideration about the distance measure is that, previously, we have defined a change identifier as returning a value proportional to the value of the property it measures at the earlier of the two dot patterns that it is passed. The distance between the locations of two patterns cannot be scaled in such a way as there is no distance value for any single pattern. Distance can, however, be made proportional to the size of the pattern.

Fig. 5.4 shows two dynamic dot patterns at two time steps. Both dynamic patterns move by the same distance but the larger pattern (Fig. 5.4(d)) is still partially within its original footprint; the impact the change has on how suitable its footprint is is less than that of the smaller pattern (Fig. 5.4(b)). The movement vector could also be made proportional to the extent of the pattern, but it is still a multi-dimensional value. There are several ways in which to approach thresholding the vector:

1. A separate identifier can be created for each of the spatial dimensions.
2. The identifier can return the multi-dimensional value scaled by extent and the framework will check each of the elements of the vector to see if they have exceeded a single value threshold.
3. The framework is set up so that multi-dimensional thresholds are allowed.

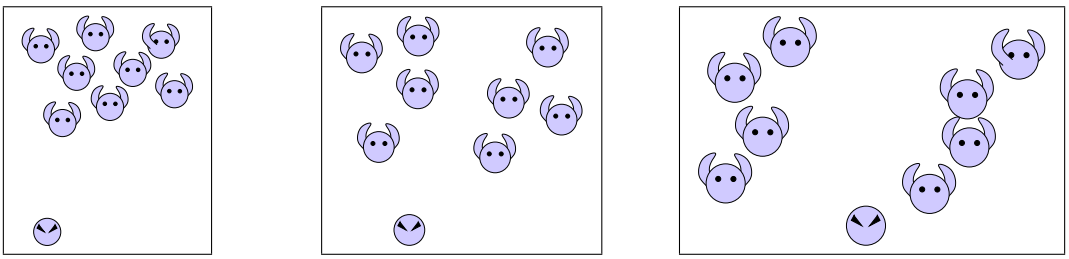
(2) and (3) both involve extensions to the framework. Such extension is not necessarily a negative, however both also add complexity when considering combining change values. We will discuss how the identifiers can be combined and why they would be in the section on change identifer sets § 5.3. (1) is an approach that avoids the problems of extending the framework or having to change the thresholding methods but it does add an extra identifier for each dimension possibly making it a slower approach. Since both the distance and the vector methods scale by extent neither fit the above definition for a change identifier. To represent this split, we call our original identifier type (identifiers that measure the difference between two descriptors that can be scaled by the measurement of that descriptor) *descriptor* change identifiers and define a *metric* change identifier as:

Definition:

A metric change identifier is a measurement that compares two phases of a dynamic dot pattern (ϕ_1 and ϕ_2) and returns, for some given metric, the distance between the patterns expressed as a proportion of some property measure of ϕ_1 such that the change identifier returns a value indicative of the impact the change has had on the suitability of the footprint.

5.3 Change Identifier Sets

We have previously mentioned that it is unlikely that a single change identifier will be able to 'catch' all forms of change. Unless it is known that a dynamic dot pattern will only change in one aspect it is likely that more than one identifier will have to be used. For example, a herd of prey fleeing a predator will change in location, extent, dimensionality and connectivity (Fig. 5.5). At the time step that the connectivity and extent changes (Fig. 5.5(b)), the location and dimensionality may not have changed appreciably. To make sure that the framework does not miss any time steps at which it should cause a footprint update, identifiers checking for change in all four aspects would be preferable.



(a) Prey have not yet noticed predator

(b) Prey have noticed predator and have started to flee; showing change in connectivity and extent (the overall location may well be the same)

(c) Prey have formed two separate components; showing change in connectivity, extent, position and dimensionality

Figure 5.5 Simplification of prey being chased by a predator

The change identifier concept allows for multiple identifiers to be used. No identifier relies on the value returned by any other and as a result they can be run concurrently[2]. Alternatively the identifiers can be ordered by importance and run consecutively. It should be noted that increasing the number of identifiers when running consecutively results in an increase in processing time, while reducing the number of identifiers reduces the amount of change types being checked for. This is not an entirely linear relation as some identifiers may be 'better' at checking and/or take longer to compute[3].

When combining multiple identifiers we must address the issue that arises from having to assign a sensible threshold for multiple types of change. As discussed earlier, this must be done in such a way that it is obvious to a user what the threshold represents and that the generality of the change identifiers has not been lost. Ideally the user should also be able to indicate that some identifiers are 'worth' more than others, i.e., that change in some identifiers is more important to the application than others.

To allow for flexibility we define a *change identifier set* as a container for multiple identifiers with properties dictating its operation. The change identifier set is a very flexible entity with properties that allow it to be fit to multiple applications. The sets used within this thesis are specified by an XML document that dictates the identifiers and the properties used (Listing 5.1).

```
1  <changeidentifierset name=''[set-name]'' ver=''[version]''>
2    <description>[Description of the set]</description>
3    <threshold>[Total threshold (%)]</threshold>
4    <maxFails>[Proportion of identifiers allowed to breach their threshold]</
         maxFails>
5    <concurrent>[Whether or not to run identifiers concurrently]</concurrent>
6    <changeidentifier>
7      <identifier>[Change Identifier Name]</identifier>
8      <priority>[Ordering, lower number = higher priority]</priority>
9      <threshold>[Identifier threshold (%)]</threshold>
10     <multiplier>[Custom user defined normalisation value]</multiplier>
11     <updateOnFail>[Whether or not to update if this
12       identifier exceeds its threshold]</updateOnFail>
```

[2]Although in running experiments using Java it was found that starting a new thread for concurrent identifiers often took longer than any individual identifier took to run.

[3]When using a single processor the same trade-off relation applies to concurrently run change identifiers.

```
13   </changeidentifier>
14 </changeidentifierset>
```

Listing 5.1 Change Identifier Set

Each set has associated meta-information (name, version and description). This information is not important to the running of the set but exists to make its identification, and therefore use, simpler. Rather than step through each XML tag, explaining its use and range of values, we will describe some possible user requirements then show an example of how the XML specification would be set up.

In the first example (Listing 5.2), our user has three identifiers (CI_1, CI_2, CI_3) that they need to run in order. If the CI_1 exceeds a change of 10% the footprint needs to be updated. If CI_1 does not exceed that threshold then only if both CI_2 and CI_3 exceed their thresholds of 10% each should an update occur.

```
1 <changeidentifierset name=''use-case-1'' ver=''0.1b''>
2   <description>Consecutive run of three identifiers</description>
3   <maxFails>0.33</maxFails>
4   <concurrent>false</concurrent>
5   <changeidentifier>
6     <identifier>CI1</identifier>
7     <priority>0</priority>
8     <threshold>0.1</threshold>
9     <updateOnFail>true</updateOnFail>
10  </changeidentifier>
11  <changeidentifier>
12    <identifier>CI2</identifier>
13    <priority>1</priority>
14    <threshold>0.1</threshold>
15    <updateOnFail>false</updateOnFail>
16  </changeidentifier>
17  <changeidentifier>
18    <identifier>CI3</identifier>
19    <priority>2</priority>
20    <threshold>0.1</threshold>
21    <updateOnFail>false</updateOnFail>
22  </changeidentifier>
23 </changeidentifierset>
```

Listing 5.2 Change Identifier Set Example 1

User number two (Listing 5.3) has twenty identifiers but has no preference about their running order. The sheer number of identifiers and the unspecified ordering make it preferable that they run concurrently. This user has no thresholds for any individual identifier but needs an update to occur if the total change exceeds 50%.

```
1 <changeidentifierset name=''use-case-2'' ver=''0.1b''>
2   <description>Concurrent run of twenty identifiers</description>
3   <threshold>0.5</threshold>
4   <concurrent>true</concurrent>
5   <changeidentifier>
6     <identifier>CI1</identifier>
```

```
7       <updateOnFail>false</updateOnFail>
8     </changeidentifier>
9     <changeidentifier>
10      <identifier>CI_2</identifier>
11      <updateOnFail>false</updateOnFail>
12    </changeidentifier>
13    ...
14    <changeidentifier>
15      <identifier>CI_20</identifier>
16      <updateOnFail>false</updateOnFail>
17    </changeidentifier>
18  </changeidentifierset>
```

Listing 5.3 Change Identifier Set Example 2

The requirement given by the second user that the *total* change not exceed 50% is worth further examination. Each identifier returns a value that can be seen as a representative of part of the total change that a dynamic dot pattern is undergoing. To produce a value for total change a method with which to combine these values is required. The combination needs to be done in such a way that we fully justify the value and are not merely creating numbers with no real-world counterpart. The concerns that we have to address when using any combination operator are:

1. Are the values combined in a way that mixes data types or scales?

2. Is undue weighting added to any value?

3. What information is lost when reducing many values to one?

The values are already normalised to percentages, as required by the change identifier definitions; so we may safely ignore (1). (2) raises a little more difficulty, if two identifiers both use descriptors from the same class then change in that class will be represented twice. For example, an identifier which measures the change in the variance from the centroid and an identifier which measures change in the area of the minimum bounding box will both indicate change if the dynamic dot pattern grows in extent. To alleviate some of this concern we have also allowed for a multiplier to be attached to each identifier. The user can mitigate (or increase) the effect of any particular identifier on the total change value by setting this property. However we strongly suspect that the multiplier would require trial and error to adjust so that the results suit the user's expectations. As discussed earlier in this chapter, this type of variable 'tweaking' is impractical even when the data is static, so with dynamic data it is likely infeasible. Instead we have looked at the measurement types (see descriptor definitions in Chapter 3) in such a way that the identifier selection process can be informed so that overlapping measurements are avoided. (2) affects any combination operator, assuming the application has cause enough to require a total threshold value then this will have to be taken into account by the user and not the framework. Question (3) is answered comparatively simply: The information lost is the manner in which change has occurred but, as for (2), if the application requires a single total value then this loss must be accepted. The simplest operator, and perhaps the most intuitive, is addition of the percentages. The three questions raise no specific objections

to the use of addition, and it is likely that any more complex method will add further confusion about what the value represents as well as increasing the processing time of the set. Hence, addition is the operator used within this thesis.

5.4 Constructing Change Identifiers

With a formal description in place the construction of some example change identifiers can be considered. We begin by providing the mathematical formula for a change identifier; this is simply the formalisation of its definition.

Descriptor Change Identifier

$$change_x(DDP_i, t, u) = \left| \frac{desc_x(DDP_i[t]) - desc_x(DDP_i[u])}{desc_x(DDP_i[u])} \right|$$

Where x is the descriptor index mapping to an element of the set of all descriptors, $change_x(DDP_i, t, u)$ is the change identifier value for the change identifier over dynamic dot pattern i at time t when the last update time was u and $desc_x(DDP_i[t])$ is the descriptor value for descriptor x on the dot pattern from dynamic dot pattern i at time t.

Metric Change Identifier

$$change_x(DDP_i, t, u) = \left| \frac{metric_x(DDP_i[t], DDP_i[u])}{desc_{s(x)}(DDP_i[u])} \right|$$

Where x is the metric index mapping to an element of the set of all metrics, $metric_x(DDP_i[t], DDP_i[u])$ is the distance between the dot patterns from dynamic dot pattern i at times t and u for the metric x and $desc_{s(x)}$ is a *suitable* descriptor for metric x by which to normalise the value.

The list of identifiers presented is by no means complete but is sufficient to provide comparison between different types of identifier and showcase some of their more interesting aspects. We will begin by listing descriptor identifiers as they are often simpler than the metric type. The identifiers are uniquely named and are formatted in small caps (i.e., ChangeIdentifier) so as to be easily referenceable.

The descriptor for a descriptor identifier must be computed for two different phases: The current phase and the phase at the timestep at which the footprint was last updated. However, the calculation at the update time has already taken place (when it was the current timestep) for all timesteps except the second (the first will always be an update time so there is no point running the change identifiers). Therefore the complexity and time taken of any descriptor identifier are equivalent to those of its descriptor.

5.4.1 Descriptor Identifiers

Cardinality

Simple to implement and run, a change identifier that measures change in cardinality is a good initial identifier. Given that this can be found while building the data structure that contains the dot pattern, we can say that, for an identifier to use it, it need not be computed and can be found in constant time.

The rest of the descriptor identifiers will be ordered as they appear in Chapter 3 apart from position which is a metric identifier. As many of the descriptors were already detailed in that chapter they will only be briefly discussed here.

Extent

VarianceFromCentroid

The centroid is the mean average position of all the dots within the pattern and as such can be found in $O(n)$ time.

$$centroid(DDP[t]) = \frac{\sum_{i=0}^{n} DDP[t][i]}{n}$$
$$variance(DDP[t]) = \frac{\sum_{i=0}^{n} (DDP[t][i] - centroid(DDP[t][i]))^2}{n}$$

Where $DDP[t][i]$ is the dot at position i in the dot pattern at time step t in the dynamic dot pattern. Variance has a computational complexity of $O(n)$ and, thus, so does the identifier VarianceFromCentroid.

BoundingBoxArea

When the descriptors were examined in Chapter 3 it was noted that this identifier ascribes a surrogate region to the dot pattern: the isothetic minimum bounding box. The bounding box can be found, even on an unordered list, within $O(n)$ time as all that is required is the extremal points in the cardinal directions of the pattern. The area for the bounding box can be found in constant time so the identifier BoundingBoxArea can also be run in $O(n)$ time. Should a tree structure be used, such as was detailed in the background chapter (Chapter 2), then the time to find extremal points becomes $O(\log n)$.

DiameterSq

The diameter is the distance between the two furthest apart dots in a pattern, and the fastest methods found in the literature to retrieve it check the vertices of the pattern's convex hull. The high complexity renders this identifier impractical to use so the approximation introduced in Chapter 3 is used (the greatest distance between the minimum and maximum dots in the plane). This identifier has a linear computation time if the data structure is an unordered list and a $O(\log n)$ time if the dots are stored in an ordered tree.

Orientation

OLSGradient
The gradient of the line found by the Ordinary Least Squares (OLS) method described in Chapter 3. OLS can be computed in linear time and is therefore well within the change identifier time requirements.

PCVectorGradient
The gradient of the vector found by Principal Component Analysis (PCA). As explained in Chapter 3, to find the Principal Component the eigenvector corresponding to the largest eigenvalue of the covariance matrix of the data must be found. Finding the eigenvalue for any given matrix is generally a complex task. However the experiments in this thesis, and the majority of the examples in the footprint algorithm and spatio-temporal literature, are 2-dimensional. The eigenvalues of a matrix are the values of k that satisfy the equation $det(A - kI) = 0$ (called the characteristic equation) in which A is the matrix, I is the identity matrix and $det(A)$ is the determinant of A [1].

$$\begin{aligned} A &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \\ det(A - kI) &= \begin{bmatrix} a-k & b \\ c & d-k \end{bmatrix} \\ &= (a-k)(d-k) - bc \\ &= ad - ak - kd + k^2 - bc \end{aligned}$$

For a 2x2 matrix the eigenvalues can be found via the quadratic formula of the characteristic equation and can therefore be found in constant time. Finding the covariance matrix has a linear time complexity, and its construction for a 2-dimensional dot pattern is:

$$cov(x,y) = \begin{bmatrix} \frac{\sum_{i=0}^{n}(x-\bar{x})^2}{n} & \frac{\sum_{i=0}^{n}(x-\bar{x})(y-\bar{y})}{n} \\ \frac{\sum_{i=0}^{n}(x-\bar{x})(y-\bar{y})}{n} & \frac{\sum_{i=0}^{n}(y-\bar{y})^2}{n} \end{bmatrix}$$

Dimensionality

EigenValueDiff
The greater the difference between the eigenvalues of the covariance matrix the greater the variance in the principal component compared to any other. In effect the greater the eigenvalue difference the closer to collinear the dots of the pattern are. Like PCVectorGradient, this identifier has a linear time complexity.

PearsonCorrCoeff
Pearson correlation coefficient can be found in linear time. Both this and SampleCorrCoeff provide measures for the interrelation between the coordinates of the dots in the dot pattern; i.e., as x increases to what degree does y change in accordance. The stronger this relation is the closer the dots are to being collinear. If the pattern is collinear and aligned to the axes then, despite the collinearity, there is no correlation between the co-

ordinates. Fortunately both PearsonCorrCoeff and SampleCorrCoeff provide a result of NaN (Not a Number) for axis-aligned patterns, therefore allowing them to be identified as distinct from patterns with a low correlation due to low collinearity.

SampleCorrCoeff
Sample correlation coefficient is also described fully in Chapter 3. We draw attention to the fact that, like the OLS method, it assumes that our knowledge of the data is not complete (hence its use of sample standard deviation). It, like PearsonCorrCoeff, can be computed in linear time so does not provide any reduction in complexity. Given that the SampleCorrCoeff and PearsonCorrCoeff are so similar, if the former outperforms the latter in identifying change in dimension then this may indicate that the dot pattern is not a true representation of the collective and has noise or missing dots.

Dispersion

Density
This is the global density of the pattern as defined by the isothetic bounding box. The complexity of finding the isothetic bounding box, as shown above, is at most $O(n)$, and, considering that the cardinality should already be known, this is therefore the maximum complexity of the Density identifier.

NearestNeighbourDistVariance
The variance of the distances between each dot and its estimated nearest neighbour (estimated for the reasons given when we examined the nearest neighbour descriptor in Chapter 3). This identifier relies on a sorted data structure to be found in a time less than $O(n^2)$. With such a data structure we can find the estimated nearest neighbour for each dot in $O(\log n)$ time, so the total time to find the variance is $O(n \log n)$. The variance of the nearest neighbour distances indicates uniformity of the spacing of the pattern, so change within it points to a change in the behaviour of the collective, for example a crowd going from calm and well spaced to panicking. The complexity is above the norm for a change identifier but it computes fast because of the limited number of computational steps required for each iteration.

Skewness
Skewness measures the tendency for the dots to be in one direction from the mean. The equation for its computation was given Chapter 3 but is repeated here for clarity:

$$s = \frac{\mu_3}{\sigma^3}$$
$$\mu_3 = \mathrm{E}[(X - E[X])^3]$$

Where σ is the standard deviation and $E[X]$ is the expected value of X, for the purposes of a dot pattern this is equivalent to $\bar{X}$. As it requires only the mean and the standard deviation Skewness can be computed in linear time.

Kurtosis

Kurtosis measures the tendency for the dots to be close to the mean. It is related to Skewness as moment about the mean, they both require only the mean and the standard deviation to be computed from iterations over the pattern, so Kurtosis can also be found in linear time.

Miscellaneous

TimeStepCount
Timestep count is the first identifier of this thesis not to have a counterpart within the dot pattern descriptors. The identifier counts the number of timesteps since the footprint was last updated and once this exceeds a set limit signifies that its threshold is broken. It functions as somewhat of a control; its purpose is to indicate whether just reducing the number of updates without any thought of measuring change is better than updating at each timestep. It serves a second role in that should it perform better than another change identifier it is likely that the 'beaten' change identifier is not measuring a useful property. To function, it 'cheats' and looks ahead to see how many phases are in the dynamic dot pattern[4]. The threshold is the percentage of phases which are allowed to occur before it forces an update of the footprint.

5.4.2 Metric Identifiers

Centroid
This identifier takes the squared distance between the centroids of the current phase of the pattern (ϕ_c) and the phase at which the footprint was last updated (ϕ_u) and divides this difference by the variance of the pattern at phase ϕ_u. Despite being a very simple identifier there is an interesting point to be made about the units it uses. The variance is the square of the standard deviation whereas the distance is a linear value. To maintain consistency of units either distance squared and variance, or distance and standard deviation should be used. Given that the process of finding the standard deviation and the distance involves first finding the variance and the distance squared, and that finding the square root is a computationally expensive operation (compared to a multiplication, for example), then using variance and distance squared makes more sense than standard deviation and distance.

BoundingBoxCentre
Bounding box cemtre is similar to the Centroid identifier, however it takes the distance squared between the centres of the isothetic bounding box of the current phase (ϕ_c) and the phase at which the footprint was last updated (ϕ_u) and divides this value by the bounding box area at ϕ_u. As has been discussed above, the time to find the bounding box is between $O(\log n)$ and n, depending on the data structure, so BoundingBoxCentre is at least as fast as Centroid, if not faster. Like with the Centroid identifier the units have been kept consistent (they are both square).

[4]This would not be possible in a live system as the data arrives with no indication of when it will end

BoundingBoxSymmetricAreaDiff

This identifier takes the symmetric area difference between the isothetic bounding boxes of the current phase (ϕ_c) and the phase at which the footprint was last updated (ϕ_u), and divides this difference by the area of the bounding box at ϕ_c. The symmetric area difference between two bounding boxes is demonstrated in Fig. 5.6, in which c is the overlap of the boxes a and b. The sum of the areas of a and b minus the twice the area of c gives the area of the symmetric area difference, shown as the shaded regions of the figure. The isothetic bounding box is used for the fast time in which it can be found, as described above, and also because it reduces the complexity of finding the symmetric area difference as there are only 8 line collisions which can occur (each line of the box at ϕ_0 can only intersect with the perpendicular lines of ϕ_1). Symmetric area difference is looked at in greater detail in the methodology chapter Chapter 6.

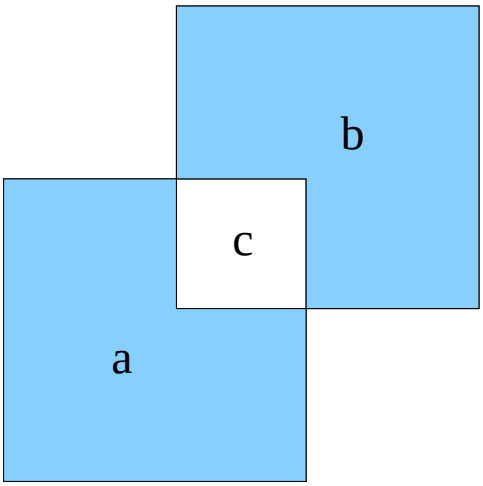


Figure 5.6 Symmetric area difference between boxes a and b. The shaded area is the difference

Identifier	Complexity	Complexity using sorted data struct.
Cardinality	$O(c)$	$O(c)$
VarianceFromCentroid	$O(n)$	$O(n)$
BoundingBoxArea	$O(n)$	$O(\log n)$
DiameterSq	$O(n)$	$O(\log n)$
OLSGradient	$O(n)$	$O(n)$
PCVectorGradient	$O(n)$	$O(n)$
EigenValueDiff	$O(n)$	$O(n)$
PearsonCorrCoeff	$O(n)$	$O(n)$
SampleCorrCoeff	$O(n)$	$O(n)$
Density	$O(n)$	$O(n)$
NearestNeighbourDistVariance	$O(n^2)$	$O(n \log n)$
Skewness	$O(n)$	$O(n)$
Kurtosis	$O(n)$	$O(n)$
TimeStepCount	$O(c)$	$O(c)$
Centroid	$O(n)$	$O(n)$
BoundingBoxCentre	$O(n)$	$O(\log n)$
BoundingBoxSymmetricAreaDiff	$O(n)$	$O(\log n)$

Table 5.2 Table showing change identifier complexities.

5.5 Performance Analysis

To establish the fitness of change identifiers for their purpose, we need to be able to measure the 'quality' of the footprint. It should be stressed that we are not commenting on how well the footprint algorithm can create a footprint that represents the pattern; we assume that the algorithm used was chosen for its suitability to the application, possibly using the footprint classification in Chapter 4. The 'quality' we measure is how close the stored footprint is to the footprint which would result if it were recomputed from the current dot pattern using the chosen algorithm at any given step. The overall quality for a sequence of dot patterns is obtained by combining the quality values for each step. Our goal is to maximise the quality while minimising the computation time. These are conflicting objectives: to maximise quality is to minimise the difference between the stored and true footprints and this can only be achieved by updating the footprint at every time step, resulting in a maximal value for the computation time. Conversely, the computation time would be minimised by never updating the footprint, typically resulting in catastrophic loss of quality. We therefore need to seek a middle course which optimises the trade-off between the objectives.

In order to compute the total time taken, we will need to make use of the following quantities:

- $t_{FP}(i)$ is the time taken to compute the footprint from the dot pattern at step i.
- $t_{CI}(i)$ is the time taken to evaluate the change identifier(s) at step i.
- $r(i)$ is a Boolean variable, set to 1 if the change identifier(s) evaluated at step i exceed(s) the pre-set threshold, and 0 otherwise.

The footprint has to be computed at least once, namely at the first timestep ($i = 0$). At subsequent timesteps it is only recomputed if the change identifiers return a value above threshold. The total computation time over a run of n dot patterns is thus

$$T_{CI} = t_{FP}(0) + \sum_{i=1}^{n} (t_{CI}(i) + r(i)t_{FP}(i)).$$

The value of T_{CI} is minimum when the change identifier threshold is set so high that the footprint is never recomputed after the start of the sequence (so $r(i) = 0$ for $1 \leq i \leq n$):

$$T_{\min} = t_{FP}(0) + \sum_{i=1}^{n} t_{CI}(i).$$

It is maximum when the change identifier threshold is set so low that the footprint is recomputed at every time step (so $r(i) = 1$ for all i):

$$T_{\max} = t_{FP}(0) + \sum_{i=1}^{n} (t_{CI}(i) + t_{FP}(i)).$$

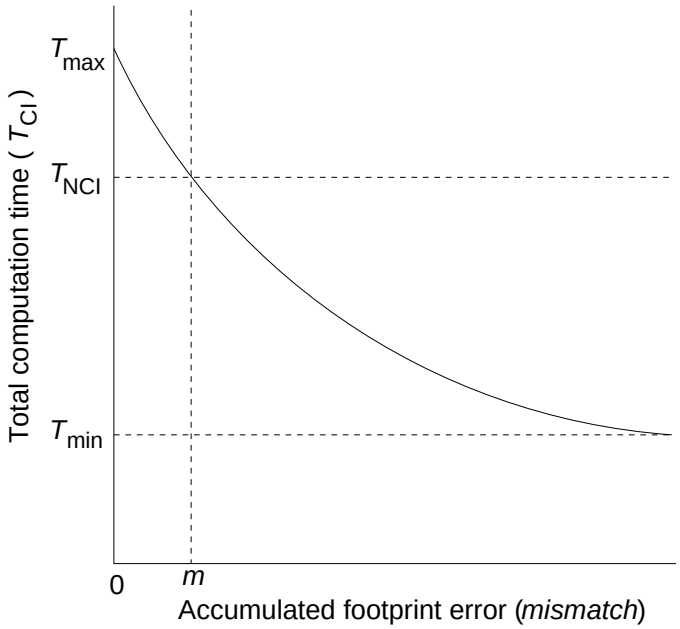


Figure 5.7 Total computation time against aggregate footprint error

If change identifiers are not used at all, and the footprint is recomputed at every timestep, then the total time taken is:

$$T_{NCI} = \sum_{i=0}^{n} t_{FP}(i) = t_{FP}(0) + T_{\max} - T_{\min}.$$

If it is assumed that always $t_{CI}(i) < t_{FP}(i)$ (for if not, there would be little point in using change identifiers) then $T_{\min} < T_{NCI} < T_{\max}$, so the relative size of T_{CI} and T_{NCI} — which provides a measure of the time advantage, if any, gained by using change identifiers — depends on the threshold settings.

It will be convenient in the following discussion to consider an inverse form of the quality measure, which we shall refer to as *error*. Our goal is therefore to seek to minimise both time and error. To measure error, we need a way of quantifying the extent of the mismatch between the stored footprint and the true footprint. The difference between two footprints can be measured in various ways, (e.g., using Hausdorff distance, or symmetric area difference) and these are discussed in Chapter 6 when the difference measure used in this thesis is explained.

If the footprint is recomputed every time, corresponding to total computation time $T_{\max}$, we have a footprint for every phase, so $mismatch = 0$. At the other extreme, the maximum value of $mismatch$ is obtained when the footprint is never recomputed, corresponding to $T_{\min}$. There is thus a trade-off between $mismatch$ and computation time, as indicated in Fig. 5.7, where different choices of change identifier thresholds correspond to different positions on the curve. The optimal setting for the change identifier threshold depends on the relative importance attached to the conflicting goals of minimizing both computation time and accumulated footprint error; but in any case no time advantage can be obtained for mismatches below the value m at which $T_{CI} = T_{NCI}$.

5.6 Summary

This chapter has provided definitions for two types of change identifier: *descriptor* and *metric* based and used these to construct several example change identifiers. How the change identifiers can be used to measure change has been examined, while paying special attention to their thresholds and how they may work in concert. The concept of a change identifier set was expanded upon and it was shown how they may be used in applications with differing requirements. Finally this chapter has presented a method by which to assess the change identifiers by the footprints that are produced when using them.

6 Methodology

Previous chapters have discussed dot patterns, footprints and change identifiers but have not yet detailed exactly how an application using change identifiers would be constructed. This chapter provides a framework in which the change identifiers can operate and shows how the experiments used in this thesis were constructed. The need to formalise the use of the change identifiers arises from the need to answer the questions that have emerged from the examinations of dot patterns, footprints and change identifiers, namely:

1. How does the dynamic dot pattern data arrive?
2. How is the data stored?
3. How is the footprint algorithm specified?
4. How are the change identifiers run?
5. How are the results displayed?
6. How can the system be tested?

The *change identifier framework* proposed in this chapter to encompass the running of the identifiers is highly modular (Fig. 6.1) in construction, allowing each of the above mentioned concerns to be dealt with individually. As shown in Fig. 6.1 the core engine of the framework requires a change identifier set and a footprint algorithm. The dynamic dot pattern is read by a buffer that 'feeds' a pattern for every timestep to the core; this pattern is processed in accordance with the change identifier set and, if an update is required, a footprint is generated using the footprint algorithm. The core sends a footprint for each timestep to the application layer (if an update has not occurred this is the same as the previous footprint) which then displays the footprint to the user.

6.1 How does the dynamic dot pattern data arrive?

When considering the way in which the dynamic dot pattern data arrives we wished to remove as many assumptions about the data as possible. The buffer (see Fig. 6.1) can be configured to work with different formats but for this thesis we use only what we consider as the bare minimum data configuration, a text file of dot locations at a given timestep with no identity information (and no guarantee that any two files have the same dot at the same position within the file). It could be argued that identity and location (or a movement vector) for the entities that have changed is less information than all of the dot

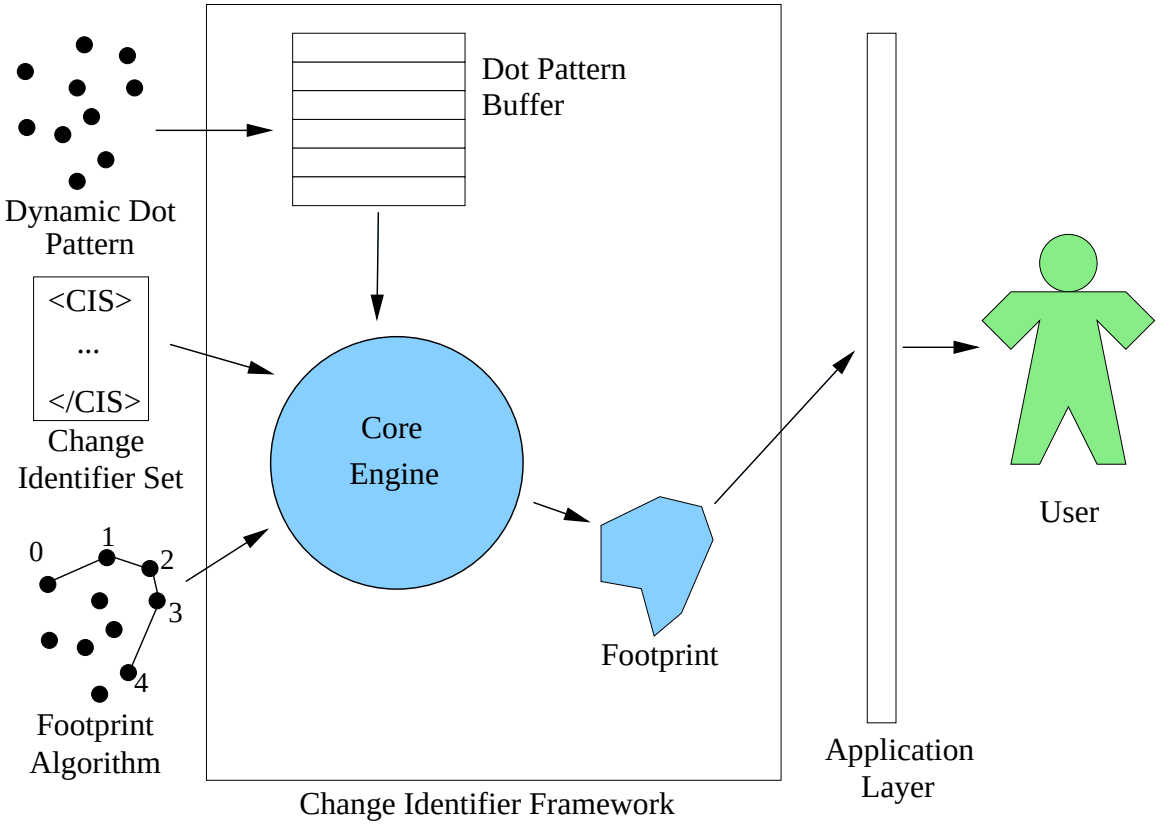


Figure 6.1 Modular Framework Architecture

locations, as it will produce a smaller file, however we took the view that reducing the types of information required would make the framework more generally applicable.

6.2 How is the data stored?

Storage of the dot pattern is a complex problem as it strongly affects the running of the change identifiers. Not having an identity associated with the dots makes performing updates on an existing structure difficult, so ideally the storage format should have a fast construction time. The main aim when considering possible data structures is to provide simple and fast access to the dots that the identifiers request. We cannot provide an optimised structure to achieve this because there is no way to know in advance all possible queries that can be made by change identifiers (there being no end to the number of identifiers that can be imagined). Instead we look at the requests that we most commonly come across in the identifiers created in for this thesis, under the assumption that these common queries are likely to be consistently occurrent across the set of all possible identifiers. What we find is that most of the identifiers only wish to sum the values of the location vectors, find the centroid, find extremal points in some dimension or find (estimated) nearest neighbours. This can be achieved by maintaining as many ordered binary trees as dimensions (one for each element in the location vector). These binary trees can be built concurrently as dots from the data file are parsed. We should note that Java (the language the framework is written in) already implements the red-black tree (Guibas

and Sedgewick [35]) in its ordered sets, however even if this was not the case, red-black trees are a suitable data structure for our purposes. As was discussed in the background chapter, the red-black tree is a binary tree (each node has at most two children) with two different types of edge: red and black, and this dichromatic approach allows it to be considered as analogous to a 2-3 B-tree (a tree that can have a up to two values at each node [11]) by thinking of the red edges as horizontal links with a black node between them. Its structure allows for easier balancing and a computationally fast insertion time without hindering its search time. As the data structure is rebuilt for each phase, the red-black tree's fast insertion and search times make it a sensible choice. The complexity for a red-black tree is $O(\log n)$ for both insert and search time in normal and worst cases. The footprint algorithms will also benefit from the small search times provided by the data structure so they are by no means being hampered when we compare the time taken for using change identifiers against the time taken to update the footprint at each phase. As a final note on data structures: If the format that the data arrives in changes drastically, the buffer being distinct from the main core of the process renders the process of changing the data structure relatively easy[1].

6.3 How is the footprint algorithm specified?

The footprint algorithm is specified at the initialisation of the program. Which would be straightforward if not for the footprint selection and parameterisation. The classification given in Chapter 4 will aid the selection of the footprint algorithm as, ideally, the user knows in advance the geometric requirements for the footprint (for example, must it be able to contain cavitities?)[2]. The parameter choice is beyond the purview of this thesis but we discuss in Chapter 10 how the identifiers might be used to help with its selection as the dynamic dot pattern changes and how the dot pattern descriptors might be used to inform the initial choice.

6.4 How are the change identifiers run?

Chapter 5 detailed how the change identifiers are described using the XML specification. The specifications are loaded into the core of the framework and the process that is implemented is shown in Algorithm 1, which works as follows: The incoming data consists of a sequence of dot patterns (e.g., from observations relayed by sensor arrays or from RFID tags attached to a flock of animals). At the beginning of the sequence a footprint $footprint(\phi_0)$ is generated for the dot pattern at phase ϕ_0 and saved as the *stored footprint* SFP_0. The phase ϕ_0 from which it is generated is stored as the *stored dot pattern* (SDP_0).

At subsequent time steps, the change identifiers are used to determine whether a new footprint should be computed; this is done by evaluating the extent to which the current phase

[1] Relative to changing the way the change identifiers are run or read in to the core

[2] The data structure may rule out some footprint algorithms that require intensity values or identities but as mentioned earlier it allows the framework to be applicable to more applications.

ϕ_i differs from the previously stored dot pattern SDP_{i-1}. If this value, $eval(\phi_i, \text{SDP}_{i-1}, \text{SFP}_{i-1})$, exceeds some pre-set threshold, then a new footprint $footprint(\phi_i)$ is generated as the new stored footprint SFP_i, and the current phase is used as the new stored dot pattern DP_i. Otherwise, the stored dot pattern and footprint are retained from the previous time step. For any phase ϕ_i, the footprint $footprint(\phi_i)$ that would be computed from it (whether or not this computation actually takes place) will be referred to (admittedly somewhat tendentiously, bearing in mind the non-uniqueness of the footprint) as the *true* footprint for that dot pattern.

Algorithm 1 Process at the Core

```
1: i = 0
2: Input first dot pattern φ_0
3: SFP_0 = footprint(φ_i)
4: SDP_0 = φ_i
5: repeat
6:    i = i + 1
7:    Input φ_i
8:    if eval(φ_i, SDP_{i-1}, SFP_{i-1}) > threshold then
9:       SDP_i = φ_i
10:      SFP_i = footprint(φ_i)
11:   else
12:      SDP_i = SDP_{i-1}
13:      SFP_i = SFP_{i-1}
14:   end if
15: until No more input available
```

6.5 How are the results displayed?

The display of the footprint is handled by the application layer. This is a necessary part of the framework for any real-world application but less so for the experiments performed in this thesis. As such the version of the program used for the experimentation runs on the command line without a Graphical User Interface (GUI). Aside from the user interface, another core difference between a testing environment and a real world application is in the consideration of the length of the dynamic dot pattern. It has been previously stated that there should be no restriction imposed on the length of the dynamic dot pattern so the framework must be constructed so that it can, theoretically, be run indefinitely. However any set of test data must come to an end and the length of the dynamic dot pattern must be known so that proper analysis can be performed.

6.6 How can the system be tested?

Over the course of the run on the dynamic dot pattern, the framework can store data that at the conclusion is passed to the test application. For example the length of time taken to process each change identifier, the time taken to process the entire time step and the change identifier that caused an update of the footprint (if any). Immediately after this

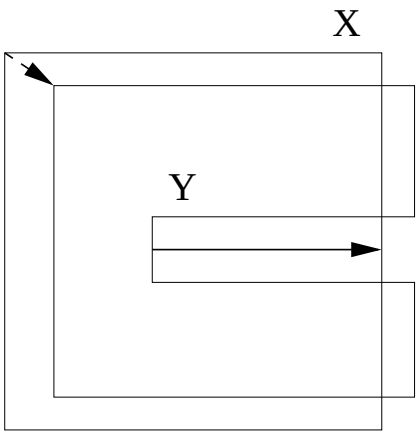


Figure 6.2 Hausdorff Distance Example

run the test application makes a call to the core for it to repeat a run over the dynamic dot pattern updating the footprint at each time step. This provides the above mentioned true footprint for each time step. As described in Chapter 5 we can use the difference between the true footprint and the stored footprint at each time step to get a measure for error. For this measure to be useful it must return a distance of 0 if the true footprint and the stored footprint are identical.

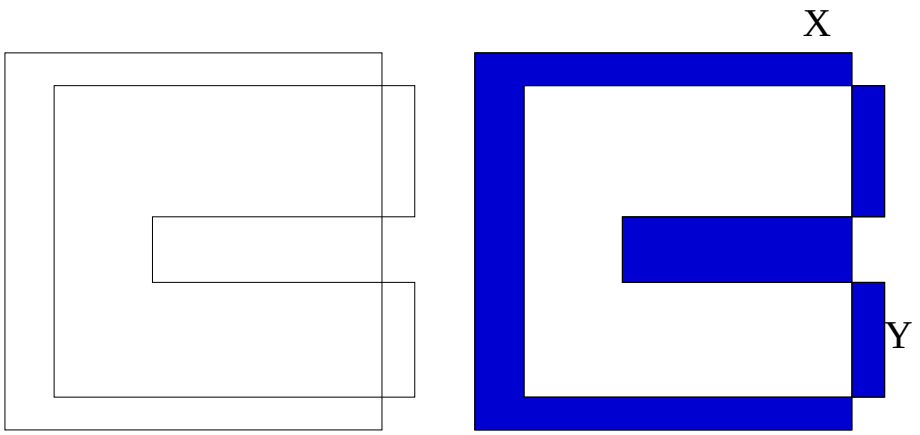
There are a number of different methods by which to ascertain the distance between two regions. Hausdorff distance, Fréchet boundary separations and symmetric area difference are three of the possible metrics that perform the measurement with different approaches ([26, ch. 7.3]). The Hausdorff distance is the greatest distance between a point within a region and the closest point in the other, Fig. 6.2 shows an example in which the greatest distance is from footprint Y to footprint X. Hausdorff distance has two variations: the Hausdorff boundary separation and the dual-Hausdorff distance. Hausdorff boundary separation is the Hausdorff distance of the boundaries of the regions and the dual-Hausdorff distance is the greatest of the Hausdorff distance of the two regions and the Hausdorff distance of the closed complements of the two regions.

Fréchet distance requires us to imagine the boundaries of the footprints as paths, then the returned distance is the distance of a line that connects the two paths at any two points. The standard illustration given of this is of a dog and its walker on the separate paths that travel at independent speeds and never go backwards; the Fréchet distance is the minimal length of leash required.

The symmetric area difference between two regions comprises the cumulative area of the parts of each region that do not overlap the other; it is given by

$$R_1 \Delta R_2 = (R_1 \setminus R_2) \cup (R_2 \setminus R_1) = (R_1 \cup R_2) \setminus (R_1 \cap R_2).$$

An example of symmetric area difference is given in Fig. 6.3, the shaded region in Fig. 6.3(b) is the parts of the regions (X and Y) that do not coincide with any part of the other region ($(X \setminus Y) \cup (Y \setminus X)$) and the area of these parts is, therefore, the symmetric area difference of X and Y.



(a) Two footprints: X and Y.

(b) The shaded region represents the symmetric area difference between footprints X and Y.

Figure 6.3 Symmetric area difference

Galton [26, ch. 7.3] provides a comprehensive discussion on these three metrics and how they relate. For now we note that symmetric area difference is the simplest to compute, as it requires only that the intersections between the edges be found. It is also an intuitively clear method for measuring the similarity of the footprints, which are areal regions, because it concerns itself with the contents of the footprint instead of its bounds. Given this simplicty and intuitive nature it is the measure we use for the experimentation performed for this thesis. Future work could examine whether different similarity measures would provide different results in the comparison of change identifiers; as long as the area of the symmetric difference provides a good measure of similarity, any difference is likely to be small. This is because the measures will only give different levels of similarity in certain specific cases (e.g., the footprint has a large external spike) and such cases are unlikely to happen consistently across the dynamic dot pattern. In effect the cases where one measure concludes that the footprints are similar and an alternative measure does not will probably average out over the run of the dynamic dot pattern.

We use the area of this as a measure of the dissimilarity between two footprints; and since we are only interested in comparisons, not absolute values, we normalise this area by expressing it as a fraction of the area of the true footprint ($footprint(\mathrm{DP}_i)$). Thus the aggregate mismatch between the stored footprint and the true footprint over a dot-pattern sequence of length n is given by

$$mismatch = \sum_{i=0}^{n} \frac{||footprint(\mathrm{DP}_i) \Delta\, \mathrm{SFP}_i\,||}{||footprint(\mathrm{DP}_i)||},$$

This *mismatch* bears a strong resemblance to the visual error used by Alani *et al.* [2]. The similarity has arisen independently and was noticed only after *mismatch* was implemented, but the concurrency adds credence to our choice of error measure.

6.7 Wasteful Processing

As an addendum to considering the methodology we note a way in which excess computation can be prevented. Many of the discussed identifiers make use of the same calculations (e.g. bounding box, centroid). It would be wasteful to perform these calculations for each identifier so a data table is attached to each time step in the dynamic dot pattern. The identifiers can query this data table, if a value does not exist then they calculate it and add it to the table for the benefit of any identifier that may require it.

6.8 Summary

This chapter has provided a modular framework for the change identifiers to be run within that allows both for real-world application use and for assessment. A data structure to contain the dot patterns has been proposed and rationalised. When considering the assessment of the change identifiers the chapter has explored three footprint difference measures and shown why symmetric area difference has been used for the experimentation within this thesis. Finally this chapter has discussed a method to reduce wasteful computation when running the change identifiers framework.

7 Results

Having examined the construction of the change identifiers, how they may be combined and used, and a methodology with which to test them in the previous chapters. This chapter discusses the parameters used within the testing, a justification on why they will be sufficient to fairly assess the change identifiers and finally presents the results from the experimentation.

7.1 The Dynamic Dot Patterns

Simply running the change identifiers over a dynamic dot pattern is not immediately possible. The first hurdle faced is in the choice or creation of the dynamic patterns. Using just one type of dynamic dot pattern will produce unreliable results, and may lead to overfitting the change identifiers to just that application type. While there is a large number of actual and potential applications which generate this data there is a dearth of available examples. Therefore, as well as the few real-world cases, we need a set of example dynamic dot patterns exhibiting a range of behaviours. A pattern generator was constructed to allow for a variety of different behaviour types including the random type used for the dot pattern analysis in Chapter 3. The dynamic dot patterns used as input for the change identifier framework are:

Real World:

- Ship tracking data[1] – Data taken by VHF tracking using the Automatic Identification System (AIS). This data set covers the English Channel for two 24 hour periods in March 2011 and shows an interesting range of movement types[2].
- Running data[3] – GPS Data that tracks runners over the Great West Run in Exeter. This set is quite small and because of the way the GPS data is collected operates over very few widely spaced timesteps.

Generated Simple:

- Translation – The dynamic dot pattern moves but has a fixed distribution from the center.

[1] Courtesy of David Walker
[2] Examples of which can be seen in [61]
[3] Courtesy of Dr Zena Wood

- Extent (Translation) – The dynamic dot pattern changes in extent by dot translation.
- Extent (Cardinality) – The dynamic dot pattern changes in extent by changes in cardinality.
- Rotation – The pattern is fixed in place but rotates around a center

Generated complex:

- Orbits[4] – A Pattern that simulates the orbit of six objects around the origin. While an application with this behaviour is unlikely to require a footprint it provides a layer of complexity to the rotation that may occur in other applications.
- Random – In which no pattern bears any resemblance to the previous pattern. Change occurs often in different ways, almost certainly requiring an update at each step. This acts as a form of control pattern to make sure that extreme changes are always 'caught'.
- Simples with noise – The simple behaviours described above with noise applied to their position or movement. This is to make sure that the identifiers can still perform even with imprecise data.
- Boid-like behaviour – Reynolds [53] identified three rules that an entity can observe which, when observed by a flock of entities, produces complex flocking behaviours:

  1. Alignment – The tendency travel in the same direction as local flockmates.
  2. Separation – Steering to avoid collision with local flockmates.
  3. Cohesion – The tendency to move toward the average position of local flockmates.

  We use a two-dimensional variation of this structure and add the concept of a variable lifespan to the entities. Having a lifespan is not necessarily to account for the possibility of the entities dying but also to allow for possibly temperamental data (in which the dots are not always reported).

The work performed by Andrienko and Andrienko [4], and Laube and Purves [46] provides a good summation of the dangers inherent in testing across limited types of dot pattern data. The wide range of types of pattern we use is such that we can avoid some of the more common pitfalls. We are, however, aware that a large proportion of our data is computer generated rather than representative of real world data, a fact we must bear in mind when drawing our conclusions.

[4]Courtesy of Dr Antony Galton

7.2 The Algorithms

The aspects of the data produced by running the identifiers in which we are most interested are the 'error' presented by the change identifiers and the time taken to run them against the time taken to run just the footprint algorithm. To be sure of a fair representation we need to use more than just one footprint algorithm. We have endeavoured to use algorithms that produce footprints in different fashions so as to give a wide range of results on which to draw our conclusions.

- Graham Scan – Graham [32] produced one of the first efficient ($O(n \log n)$) algorithms for finding the convex hull. The paper [32] is a succinct mathematical description of the algorithm, so to avoid repition we simply note that it expresses all the dots of a pattern in polar coordinates from an origin point known to lie within the pattern, and from this origin successively removes points that cannot be extremal.

- α-shape – As described in Chapter 2 the α-shape is the conjunction of all circles of radius $1/\alpha$ that contain either all/none[5] of the dots of the pattern. Choosing a parameter that would always produce footprints in-between the convex hull and the null footprint is not easy (rather tentatively called 'interesting' footprints for the sake of brevity). The average estimated nearest neighbour distance[6] was used as a likely candidate.

- χ-hull – The χ-hull (Duckham *et al.* [20]) relies on the Delaunay triangulation. Using a divide and conquer method this can be found in $O(n \log n)$ time and the χ-hull process then removes lines of length greater than the given parameter when removing them does not 'break' the hull. Like α-hull a parameter value needs to be chosen that will produce 'interesting' footprints. The parameter also has a direct effect on the process time of the algorithm, the smaller the line length the more lines there will be to check to see if they can be removed. The average estimated nearest neighbour distance provides a value that fits these requirements[7].

- Descending Swinging Arm – The Swinging Arm algorithm (Galton *et al.* [28]) can be extended to function as an iterative process in which successive arm lengths are tried. The descending version begins with an arm length equal to the diameter of the pattern, a value guaranteed to give the convex hull. Then by choosing the second longest side length of the footprint the arm length decreases until the footprint contains a degenerate line, at which point the footprint previous is selected.

- Ascending Swinging Arm – This version of the Swinging Arm algorithm starts with the average nearest neighbour distance and increases in increments of the shortest nearest neighbour distance[8] In comparison to the descending swinging arm al-

[5] All for positive α and none for negative

[6] Multiplied by -1 to give a α-shape with concavities.

[7] It should be noted that we are not suggesting that the average nearest neighbour distance always provides a 'good' footprint but that it tends to produce an 'interesting' one

[8] Or by a quarter of the average if the shortest is less than that value. This is done so as to reduce to the number of iterations.

gorithm, the ascending version is more likely to produce footprints with multiple components.

- Minimum Isothetic Bounding Box – The dual red-black trees allow us to find the extremal dots in the x and y coordinates in $(\log n)$ time. It is likely, therefore, that the minimum isothetic bounding box can be computed in less time than some of the change identifiers. It is included to make sure that any positive bias towards change identifiers from the implementation of any of the above algorithms is balanced.

7.3 The Change Identifier Sets

As this chapter is looking to make assessments on the general ability of change identifiers to reduce footprint updates, there will need to be a range of change identifier sets. To provide this range several sets are created: A set for each of the identifiers introduced in Chapter 5 as individuals, a set for the change identifiers based on the fastest descriptors from Chapter 3 (i.e., Skewness, Kurtosis, PearsonCoeffCorr, OLSGradient, Cardinality and DiameterSq) a set for the fastest descriptor change identifiers in conjunction with the metrics from Chapter 5, and a set containing all of change identifiers. For each of these twenty sets a version is created with the total thresholds 0.1, 0.25, 0.5, 0.75 and 0.9 giving a total number of change identifier sets used of one hundred. With a hundred different change identifier sets there should be enough scope to see sets that reduce the number of updates for a suitably small trade-off in error.

7.4 Plotting the experiments

Given a set of dynamic dot patterns, a set of footprint algorithms and a collection of change identifier sets we need to decide on an appropriate, defensible procedure with which to test them. We will call each pattern, footprint and change identifier set combination a *run* of the framework; each run produces a graph of time taken with change identifiers and without against timesteps (7.1) and a graph of error against timstep (Fig. 7.2). For brevity we shall refer to the run over the dynamic dot pattern when using change identifiers to signal footprint updates as **with** (i.e., *with* change identifiers) and the run when updating the footprint at each time step as **without** (i.e. *without* change identifiers).

The graphs shown in Fig. 7.1 and Fig. 7.2 are of the same hypothetical experiment, as evidenced by the correlation of their 'spikes'. Each update of the footprint, when using change identifiers has a 'jump' on the time taken graph (Fig. 7.1) and a corresponding 'drop' to 0 on the area difference graph (Fig. 7.2). The 'jump'-height is the time taken to run both the change identifiers and the footprint algorithm, and the 'drop' to 0 indicates that the footprint from the change identifier run is identical to the 'true' footprint. Fig. 7.2 show two types of slope that represent different changes in speed in the dynamic dot pattern: *gradual* and *steep*. The phases in the timestep range τ_4–τ_8 show a gradual increase in the error, represented by the symmetric area difference, until timestep 9 at

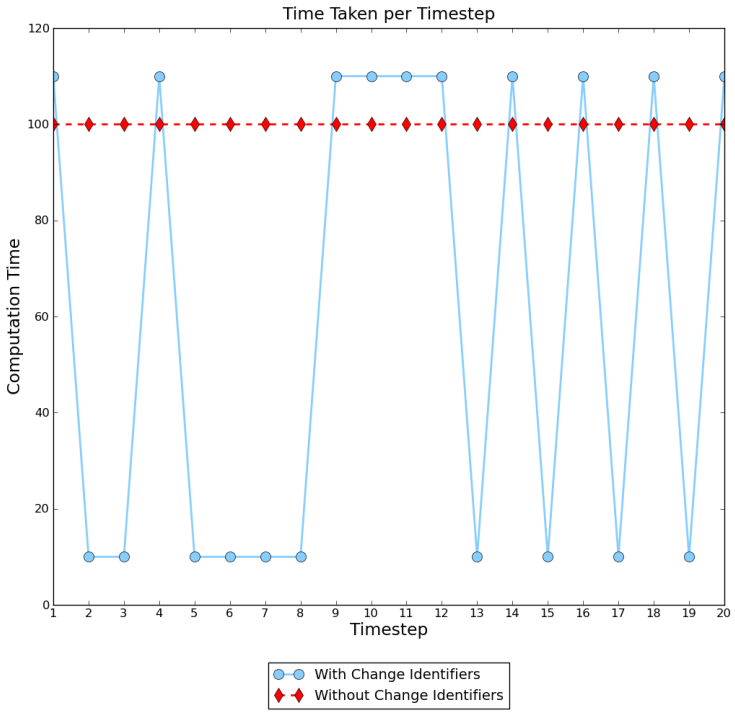


Figure 7.1 Example of a graph of time taken per timestep.

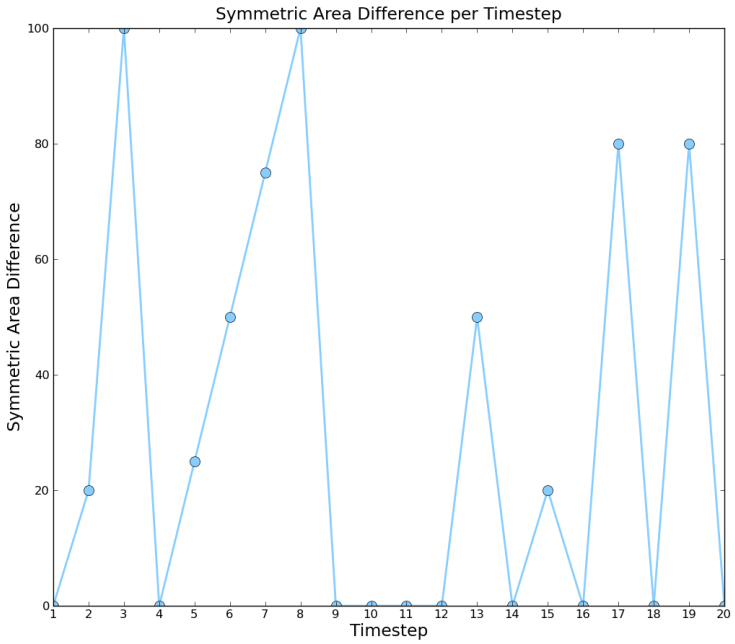


Figure 7.2 Example of a graph of symmetric area difference per timestep.

which point the thresholds were exceeded on the change identifier set that was being used. The steep slopes between τ_{16}–τ_{20} are indicative of much faster change than in the τ_4–τ_8 range as the symmetric area difference increases far more between each timestep. Further to showing the rate of change, the symmetric area difference can also denote different change behaviours in the dynamic dot pattern: *uniform* and *erratic*. Both the ranges

τ_4–τ_8 and τ_{16}–τ_{20} show uniform change in the dynamic dot pattern as the increase in symmetric area difference is constant but between timesteps τ_{12} and τ_{16} the symmetric area difference follows the erratic sequence $\langle 0, 50, 0, 20, 0 \rangle$, as the graph shows that a symmetric area difference of 100 can be reached before an update occurs, there must be large changes at τ_{14} and τ_{16}.

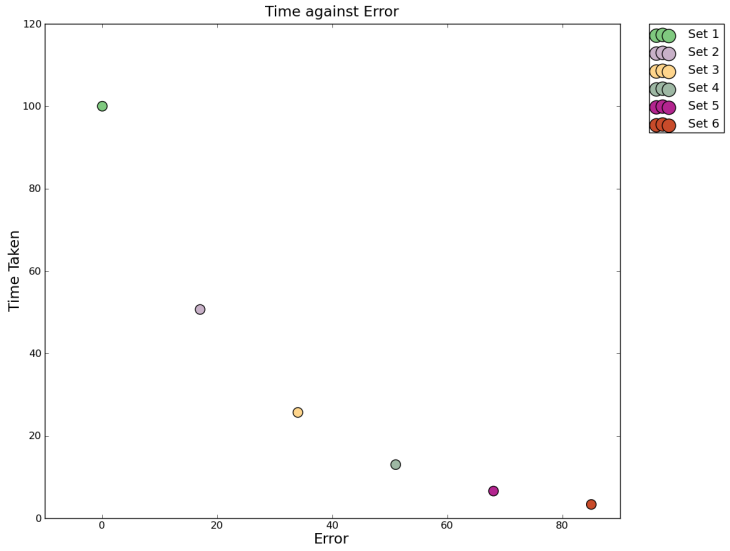


Figure 7.3 Example of a graph of time against error for multiple change identifier sets.

By iterating over the change identifier set collection we also fill in points on a graph of total error against total time taken Fig. 7.3. Each point on Fig. 7.3 represents the performance of that change identifier set on the given dynamic dot pattern for the given footprint algorithm. Altering the algorithm will not change the performance of the identifier sets relative to each other but it will affect the values given to their performance. For example the time differences between **with** and **without** will be far greater for the Ascending Swinging Arm algorithm than for the Graham Scan algorithm. By taking an average performance score over the set of footprint algorithms we can be sure that we are neither penalising the change identifiers nor being overly biased towards them. The results can be generalised further by taking an average performance score over the set of different dynamic dot pattern types. With these considerations in place we can make confident statements about the performance of a change identifier set in general as well as for a specific dynamic dot pattern type (e.g., A change identifier set may perform particularly well when the pattern changes only by extent but terribly for a dynamic pattern which rotates.)

7.5 Results

There are one hundred identifier sets running over twenty-one dynamic dot patterns using six footprint algorithms leading to $12,600$ individual runs. Each run has values for total time taken, total symmetric area difference (error), average time taken per timestep, av-

erage symmetric area difference per timestep and average symmetric area difference as a proportion of the area of the true footprint per timestep. With the sheer amount of data it is impractical to plot all the results on a single graph so many of the following graphs show only the best performing thresholds for each set and with their values averaged over the footprint algorithms.

7.5.1 Footprints

Before showing the results of the change identifiers, the footprint algorithm times are graphed so as to provide context for the times taken when performing **with** runs. Fig. 7.4 shows the average time taken per timestep to run each footprint algorithm for each of the dynamic dot patterns. The α-shape algorithm consistently takes longer to compute than the others; it seems likely that this high computation time is due to the iterative construction process of the algorithm that contains regular distance checks (to see if any dots fall within the discs that are defined by its candidate edges). The expectation would be that the minimum isothetic bounding box, being the computationally least complex, would be the fastest to compute and this is shown in the data. We note that the computation time appears to be almost constant but this is an effect of the logarithmic scaling and disappears when the bounding box computation time is looked at by itself (Fig. 7.5). This thesis does not focus on the footprint algorithms and their differences, but future work could look at answering such questions as: Why does the Ascending Swinging Arm tends to compute faster than the Descending variant? And why does the χ-hull only perform quicker than the both of the Swinging Arm variants when the dynamic dot pattern has featured expansion by increasing cardinality?

7.5.2 Change Identifier Sets Time

Fig. 7.6 is a time taken graph for the change identifier sets in which only the thresholds with the minimum time are plotted for each set; the red diamond indicates the average time taken for a **without** run to be performed on the dynamic dot pattern. The thresholds are all 0.75 or 0.9 and therefore to the high end of the range that was specified. This is expected as the higher the threshold the less likely the change identifiers are to cause an update. The reason they are not all at the 0.9 threshold is that, for those with a 0.75 threshold, the number of footprint updates for both runs using 0.75 and 0.9 are the same and therefore they have the same computation times; the lowest threshold is plotted as a default. Important to note on Fig. 7.6 is that all the change identifier sets take less time than the **without** run.

7.5.3 Change Identifier Sets Error

Unlike Fig. 7.6, Fig. 7.7 plots the average error (measured with symmetric area difference) per timestep instead of the time taken. The graph shows the thresholds for each identifier

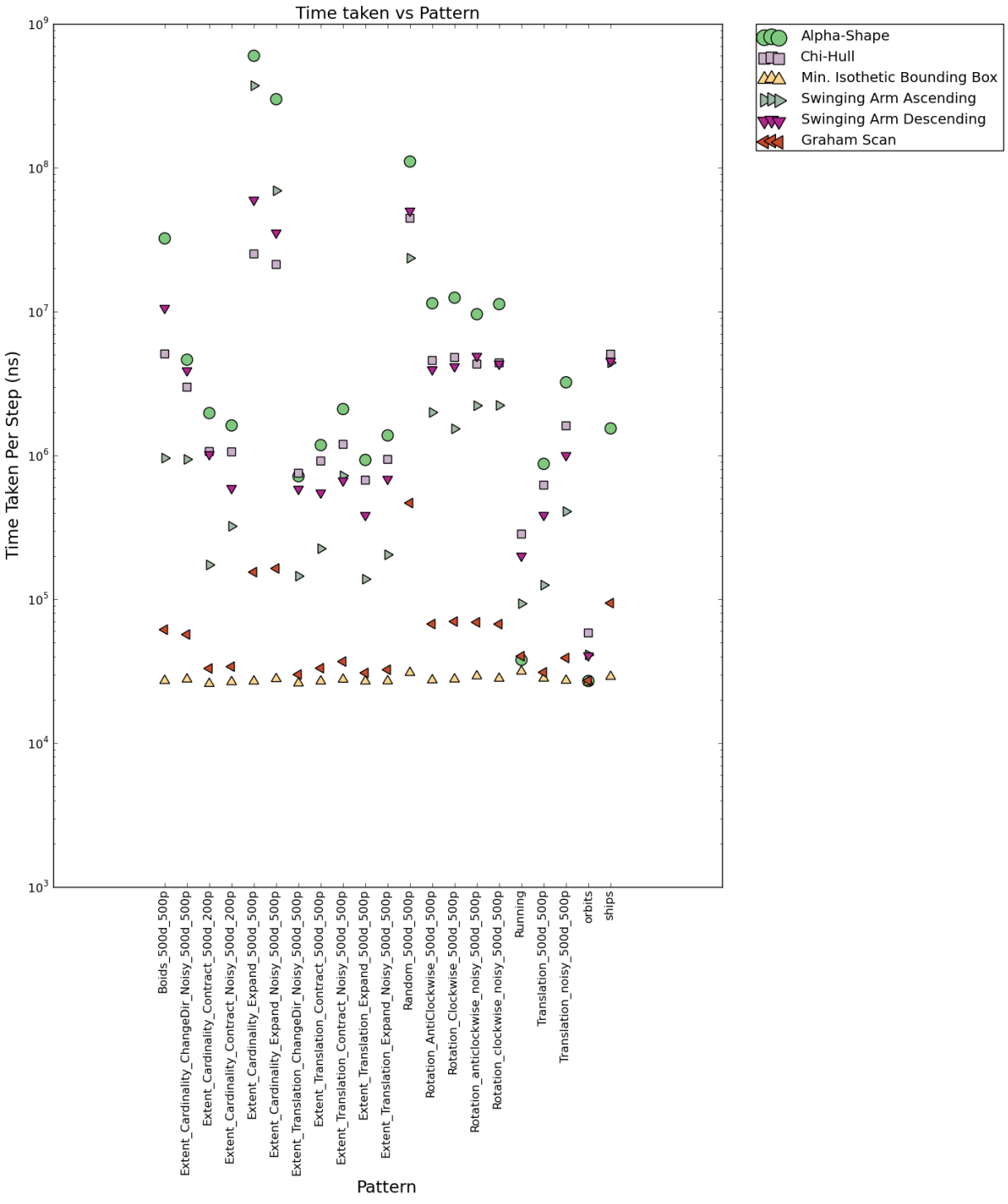


Figure 7.4 Average time taken per timestep for each footprint algorithm on each dot pattern type.

set that minimised the error; all of which are 0.1. The lower the threshold the more likely the change identifier set is to cause footprint updates, and the more footprint updates the less the symmetric area difference between the stored footprint and the 'true' footprint. A final point of interest on this graph is that the sets do not appear in the same order for every dynamic dot pattern type indicating, the perhaps intuitive fact, that different dynamic patterns change in different ways and that an identifier that can catch one type of change will not necessarily catch another.

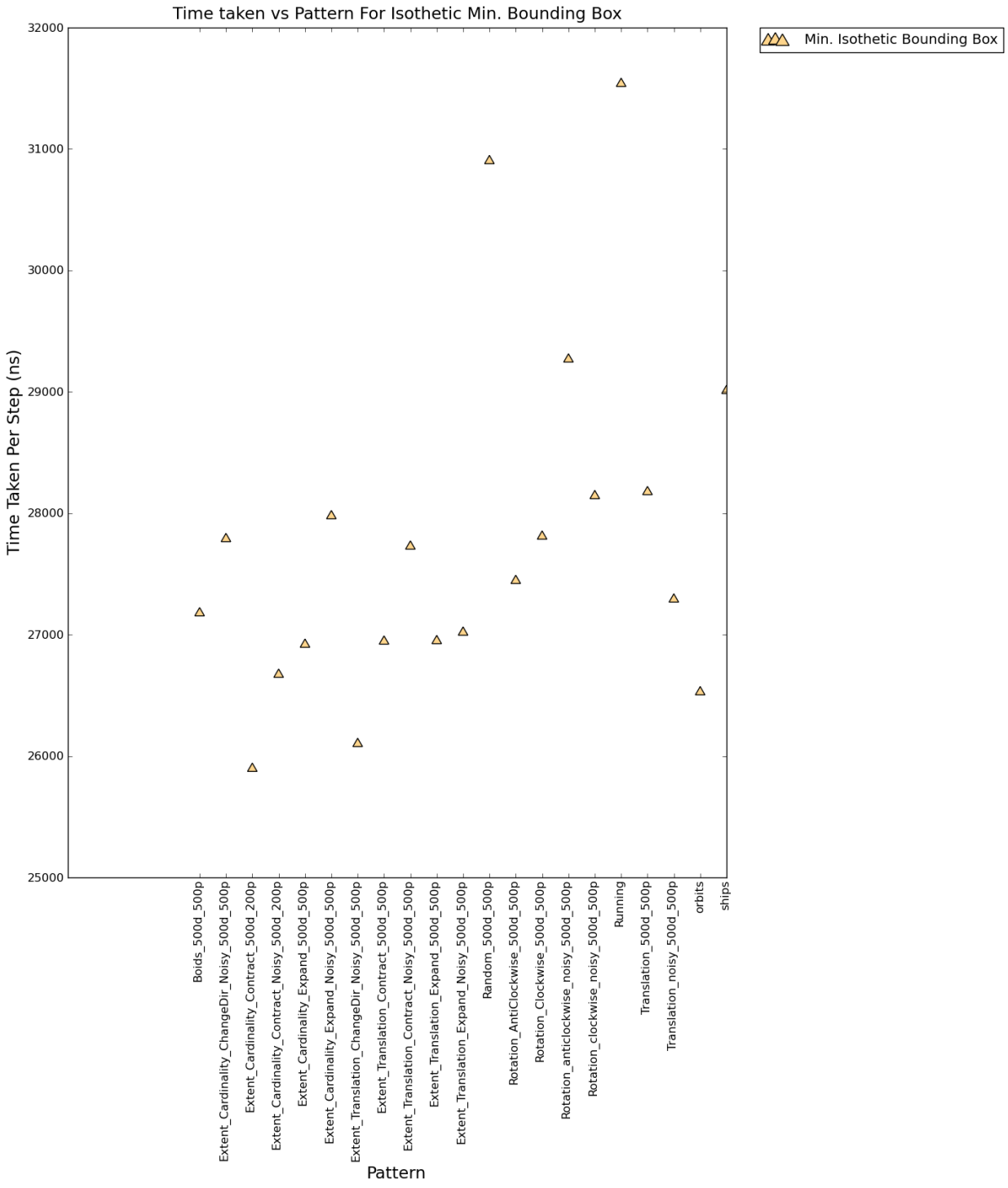


Figure 7.5 Average time taken per timestep for the isothetic minimum bounding box algorithm on each dot pattern type.

7.5.4 Time against Error

The error and time against dot patterns graphs are useful for the conclusions that have been drawn from them, but they do not allow for an assessment of change identifiers that takes into account both their time taken and their error. Figures Fig. 7.8 and Fig. 7.9 show the average time taken per timestep against the average error per timestep, and the

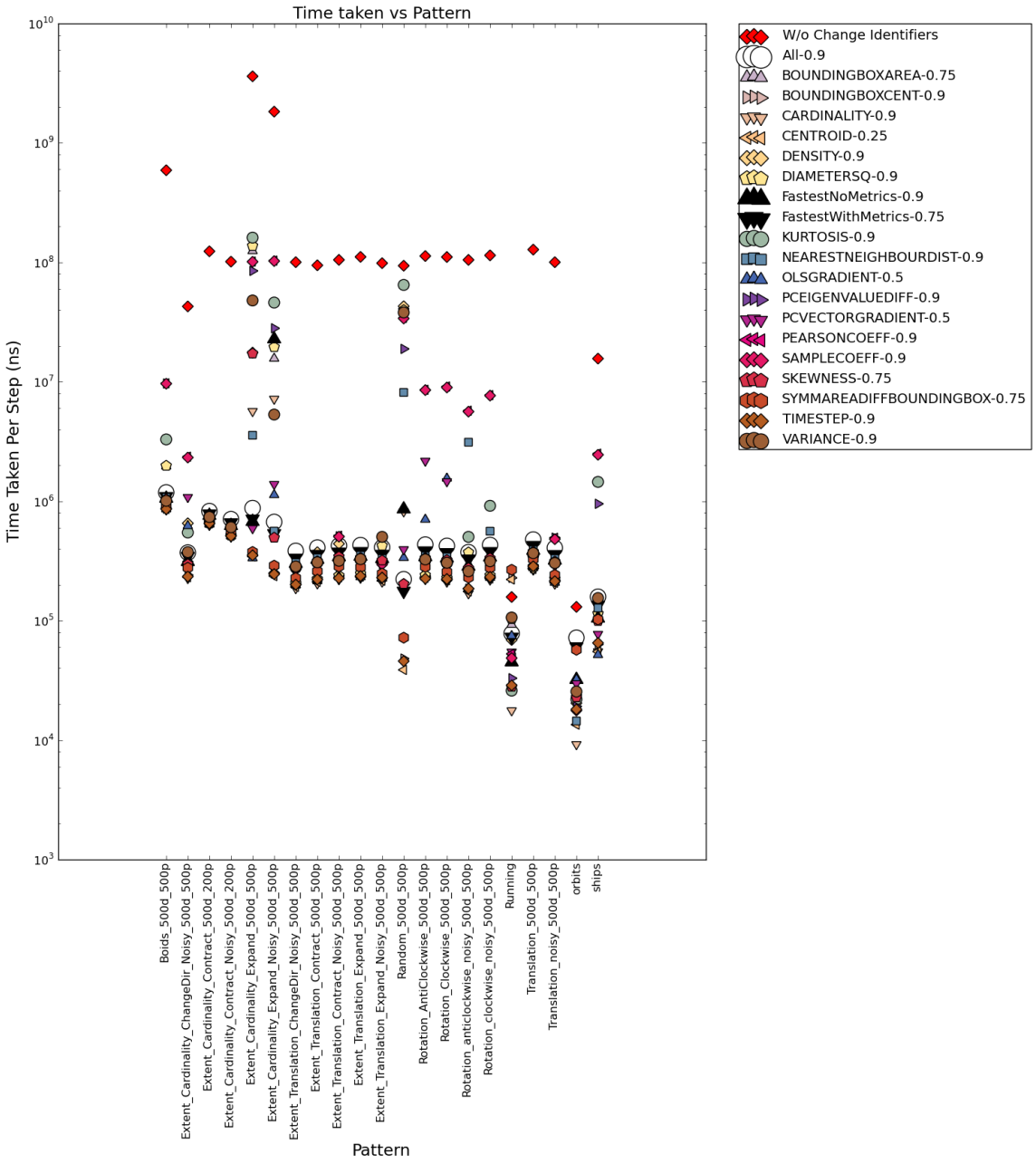


Figure 7.6 Average time taken per timestep for the fastest performing thresholds of each change identifier set on each dot pattern type.

total time taken for the run against the total error for the run respectively for the best performing change identifier set thresholds. Best performing in this case are the thresholds which perform better than their contempories on at least one of reducing the time taken or reducing the error. This form of comparison uses the idea of dominance, which will be fully explained in Chapter 8; for this chapter we simply note that any change identifier set threhold not plotted made no improvement on those that were in either time taken or error. The symmetric area difference as a proportion of the true footprint's area is not, yet, used as we are only comparing sets against each other and not evaluating the performance of a set. The values in the graphs represent the average result over all dynamic dot patterns and all footprint algorithms, and the blue line in each graph indicates the time taken to

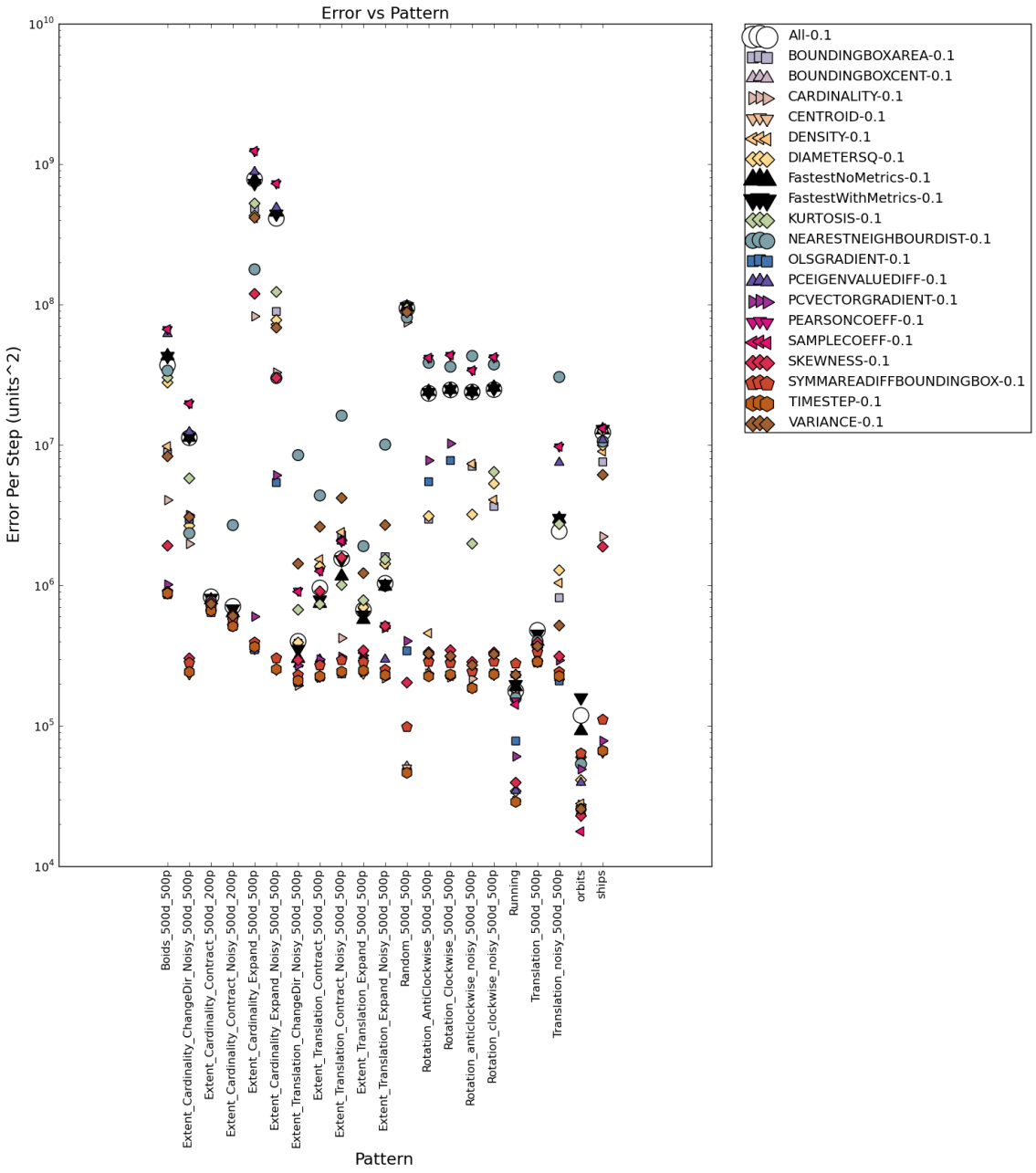


Figure 7.7 Average error per timestep for the best performing thresholds of each change identifier set on each dot pattern type.

complete a **without** run (this **without** run is analagous to the the T_{NCI} line from the trade-off graph Fig. 5.7 in Chapter 5).

The change identifier sets have formed definite bands in both graphs. The bands have been enlarged in the subgraphs of each figure so that they may be better examined. The clear banding differences are an indication that there are set points in the dynamic dot patterns at which not updating causes large increases in the symmetric area difference.

The only significant difference beteen Fig. 7.8 and Fig. 7.9 is that the figure showing totals (Fig. 7.9) has concatenated Bands 3 and 4 from the figure showing averages per timestep (Fig. 7.8) into a single Band 3; this is likely an artifact of the scaling differences between

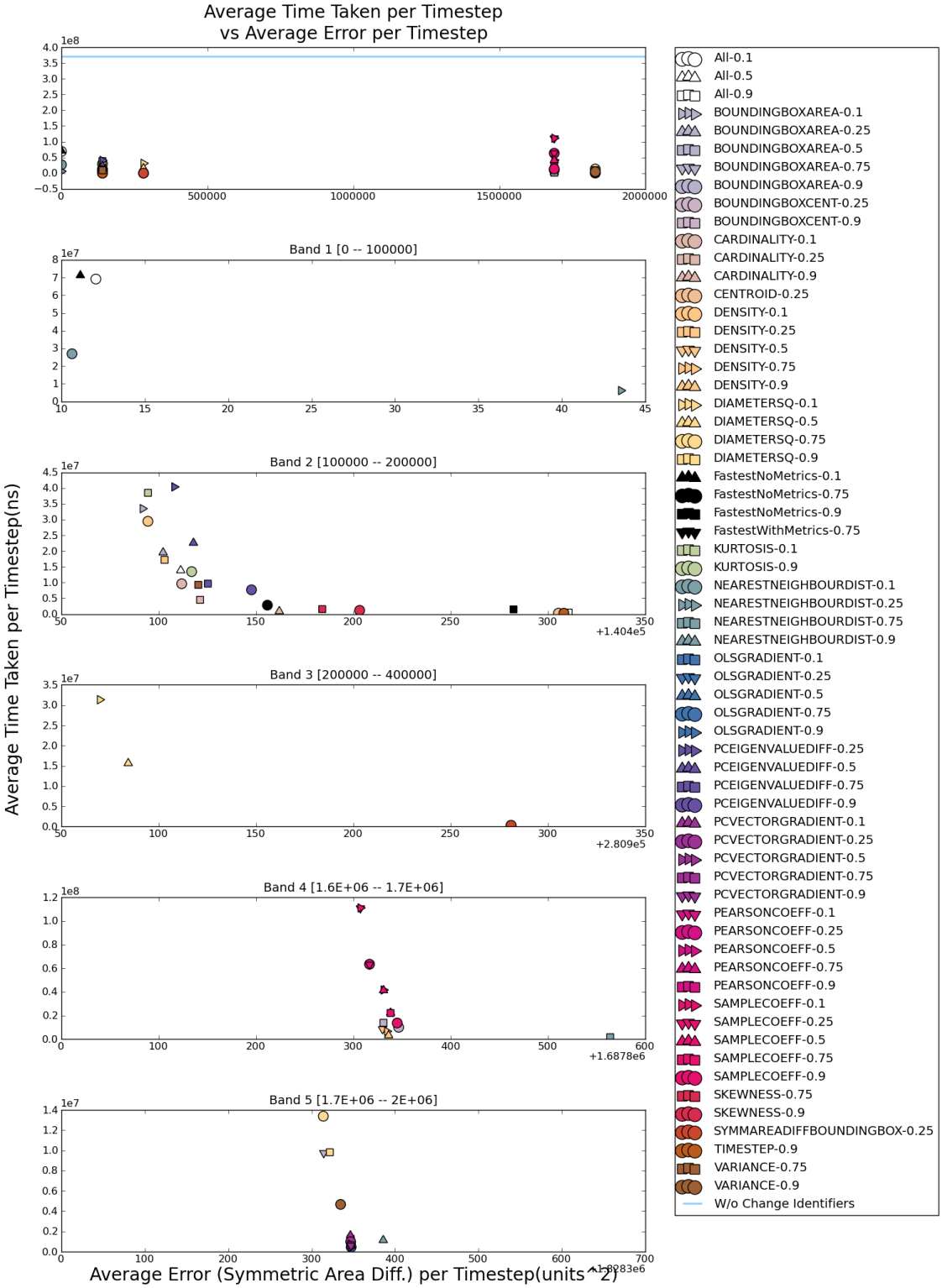


Figure 7.8 Time taken against error per timestep averaged across all dot patterns for the best performing thresholds of each change identifier set

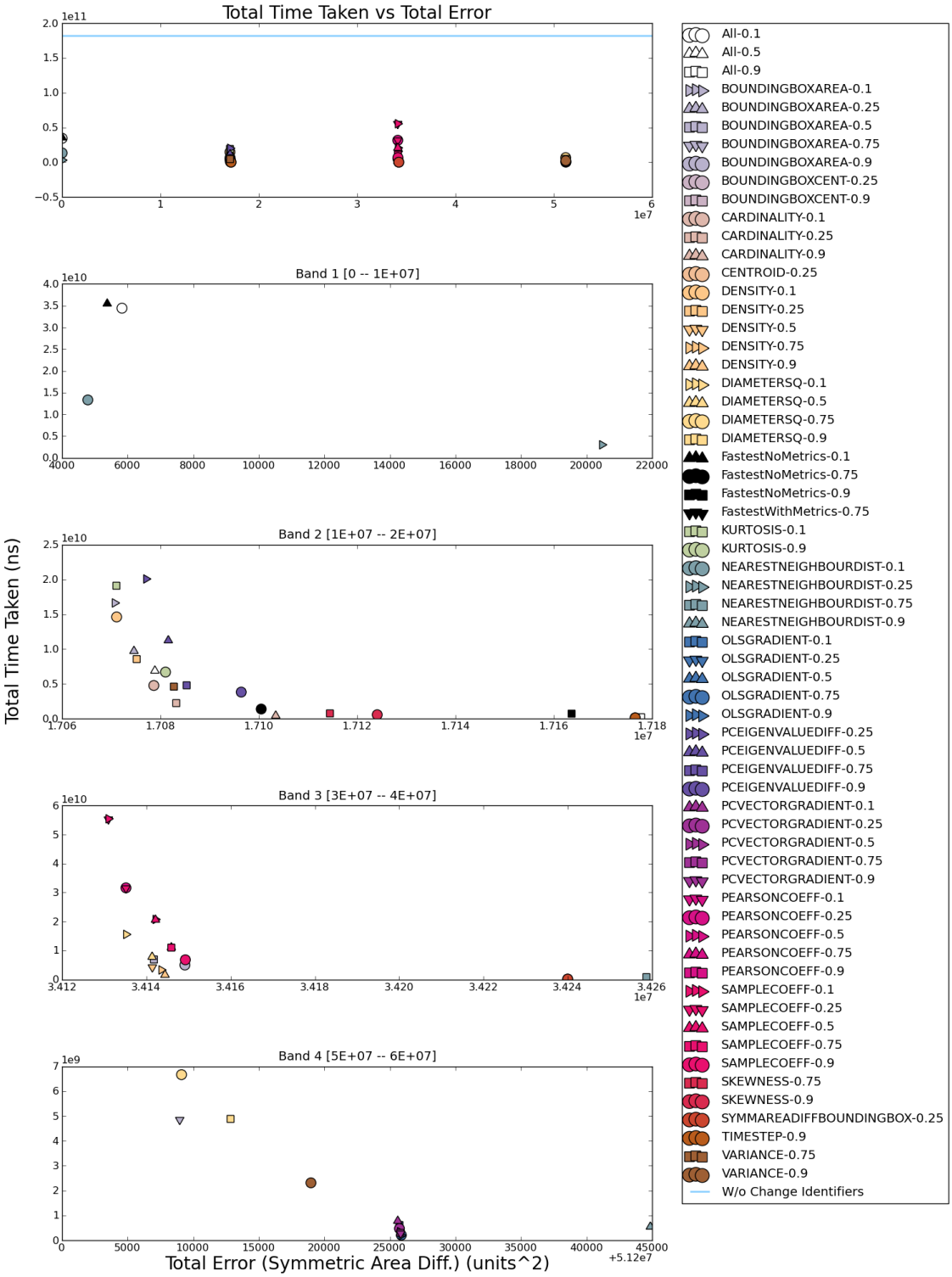


Figure 7.9 Time taken against error totalled for each run averaged across all dot patterns for the best performing thresholds of each change identifier set

total and average. On both figures Band 2 has a good example of the time-error trade-off that was discussed in the change identifiers chapter (Chapter 5), in fact all the bands have an indication of this curve but none so strongly realised as in the second band.

All of the identifier sets are well below the computation time when running **without** so we focus on the first band, in which the identifiers have minised the symmetric area difference. The identifier sets appearing in Band 1 have been re-plotted (with all their best performing threshold values) onto Fig. 7.10[9]. We note that once again there is an indication of the time-error trade-off curve appearing. The thresholds of each identifier place them where we would expect on the curve with the lower thresholds to the top-left and the higher thresholds to the bottom-right giving credence to the statement that the trade-off can be traversed by changing the threshold value.

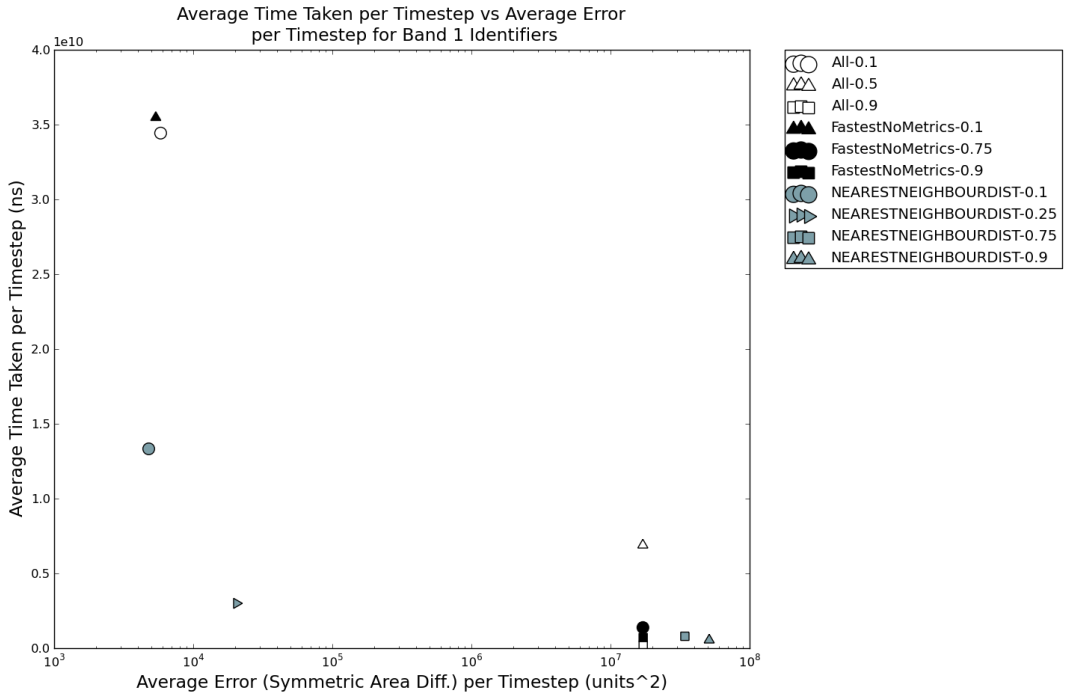


Figure 7.10 Time taken against error per timestep averaged across all dot patterns for the best performing thresholds of the identifiers appearing in Band 1

A final note on the time against error graphs is how well the NearestNeighbourDistVariance identifier (estimated nearest neighbour distance variance) has performed, with its lowest threshold value producing the least error score. This does not mean that NearestNeighbourDistVariance is a 'better' identifier than any other necessarily but, given the averaging across the twenty-one different types of dynamic dot pattern, suggests that it is a very generally applicable change identifier.

[9]The graphs are only of the average timestep results and not the total results to reduce clutter as both are very similar; differing in only the scales

7.5.5 Specific Runs

The previous results have all focused on the results averaged across the dot patterns and/or the footprint algorithms for multiple change identifiers. In this section we will examine the results from individual runs of an identifier.

As NearestNeighbourDistVariance-0.1 performed so well across the averages we will look primarily at its results. The first two figures (Fig. 7.12 and Fig. 7.11) are of runs that used the dynamic dot pattern that expanded in extent via change in cardinality with noise, as it presented the greatest variance in both change identifier time and error, and the χ-hull, as it had the median performance on that same dynamic dot pattern.

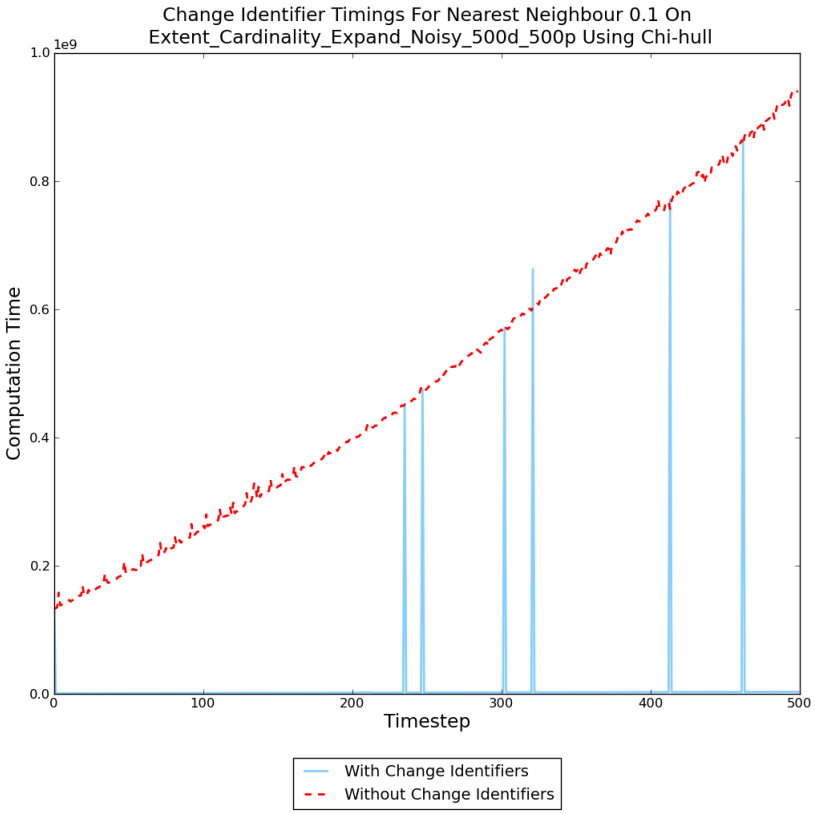


Figure 7.11 Time taken to compute at each timestep for NearestNeighbourDistVariance–0.1 on the Extent-Cardinality-Expand-Noisy dynamic dot pattern using the χ-hull footprint algorithm.

Fig. 7.11 shows that, even as the computation time for the pattern increases with the cardinality, the change can still measured within far less time than the algorithm takes to run. Fig. 7.12 displays the symmetric area difference at each step, however the symmetric area difference is not a good measure of how well the footprint at any timestep has matched the true footprint. To clarify, the greater the similarity between the stored footprint and the true footprint the lower the symmetric area difference will be, but this does not indicate what a 'good' value of symmetric area difference for a pattern is. Fig. 7.13 shows an example of this effect; both box pairs A and B have the same symmetric area difference, however the box b_2 could well be considered a better fit to b_1 than a_2 is to a_1. To be able to draw conclusions about how well the footprint has been maintained the equation for *mismatch* from the change identifiers chapter is used. The symmetric area difference

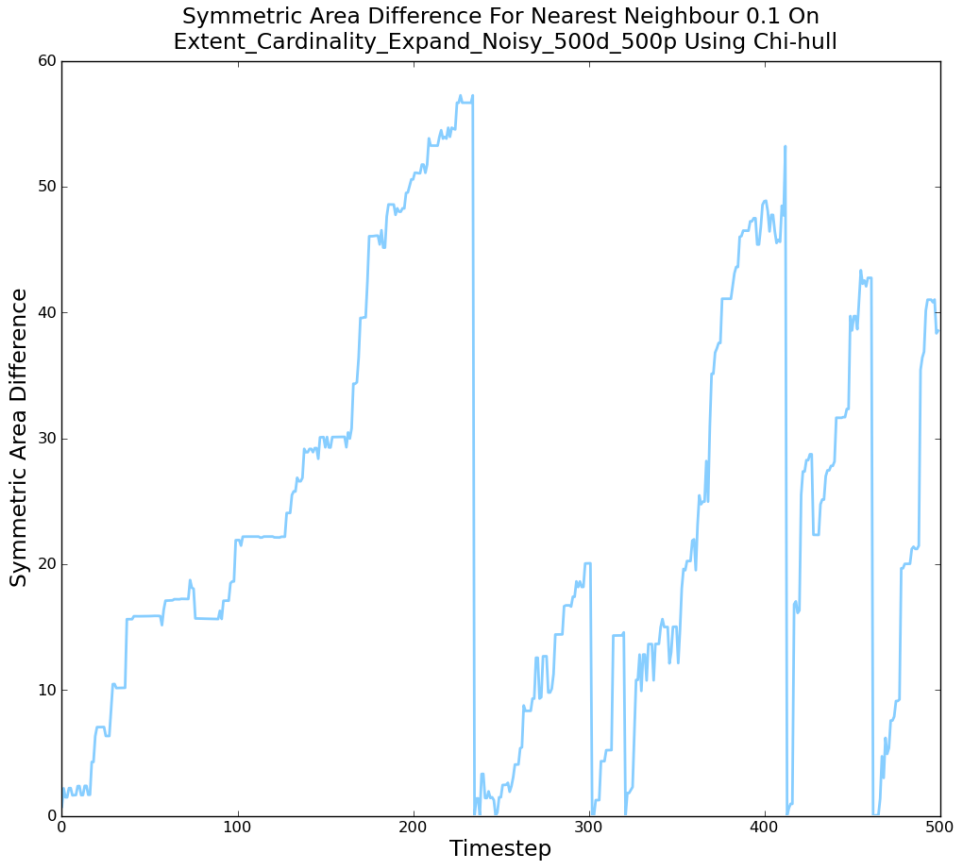


Figure 7.12 Symmetric area difference at each timestep for NearestNeighbourDistVariance – 0.1 on the Extent-Cardinality-Expand-Noisy dynamic dot pattern using the χ-hull footprint algorithm.

at each timestep is taken as a proportion of the true footprint, an example of which is shown in Fig. 7.14. The maximum proportionate area difference that is reached is less than 2.5×10^{-7} units^2, and we can say, with some confidence, that a difference so small demonstrates that the true footprint is being well tracked.

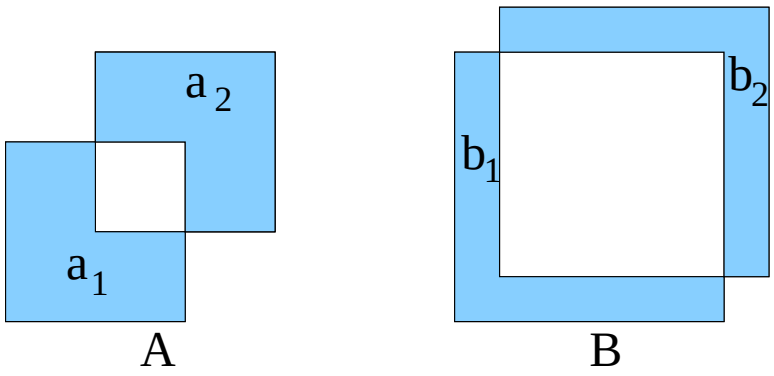


Figure 7.13 Example of identical symmetric area differences for different sized footprints

To be certain that the choice of the χ-hull is not unfairly weighting the graphs in the change identifiers favour we have also included the run graphs from the same dynamic dot pattern but using the isothetic minimum bounding box as the footprint algorithm. Even with the low complexity of the bounding box algorithm Fig. 7.15 demonstrates that some change identifiers can still operate in less time. Fig. 7.16 matches the Fig. 7.14

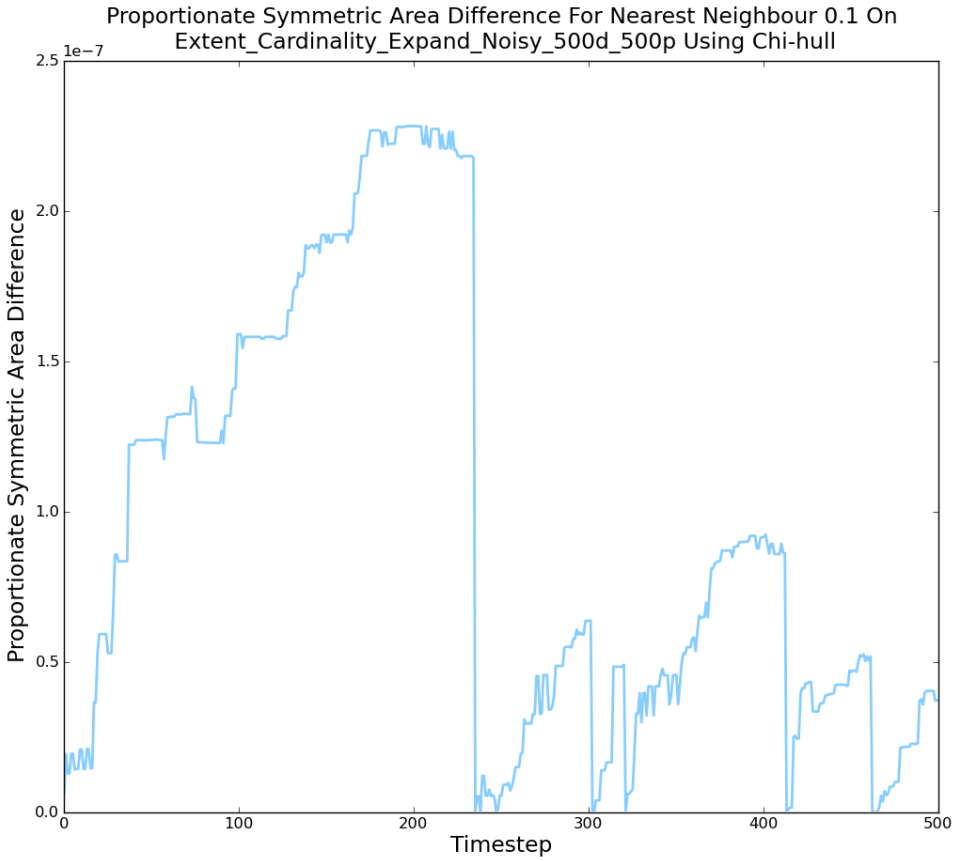


Figure 7.14 Proportionate symmetric area difference at each timestep for NearestNeighbourDistVariance – 0.1 on the Extent-Cardinality-Expand-Noisy dynamic dot pattern using the χ-hull footprint algorithm.

but is somewhat more 'blocky' as would be expected when dealing with symmetric area differences on the minimum bounding box.

The dynamic dot pattern used in the above graphs has a relation between each phase. While it is noisy, the pattern changes in a consistent fashion and this is evidenced in the steady increase of symmetric area difference shown in Fig. 7.12, Fig. 7.14 and Fig. 7.16. Within the set of dynamic dot patterns used for this thesis there is a random pattern in which no phase need bear any relation to the previous. We would not expect change identifiers to function particularly well for a pattern that requires such regular updates. However, should the dynamic dot pattern have brief respites in which the difference between two phases is not great it will be possible for change identifiers to save some computational time. Fig. 7.17 shows the time comparison for **with** and **without** runs on the random dynamic dot pattern using the χ-hull footprint algorithm and is, due to the many footprint updates required, decidedly cramped. Fig. 7.18 shows the times taken for the same experiment as the difference between the **without** and **with** runs. The line stays mostly above the 0 point on the y-axis, and we can therefore state that, even if it is only by a small amount, the change identifiers are still saving time. With regard to the error when running **with** on the random dynamic dot pattern we look at the graph of proportionate symmetric area difference: Fig. 7.19. The graph has been cropped because the scaling made it difficult to see the timesteps at which updates occurred; the difference between the symmetric area differences ranging between several orders of magnitude. For

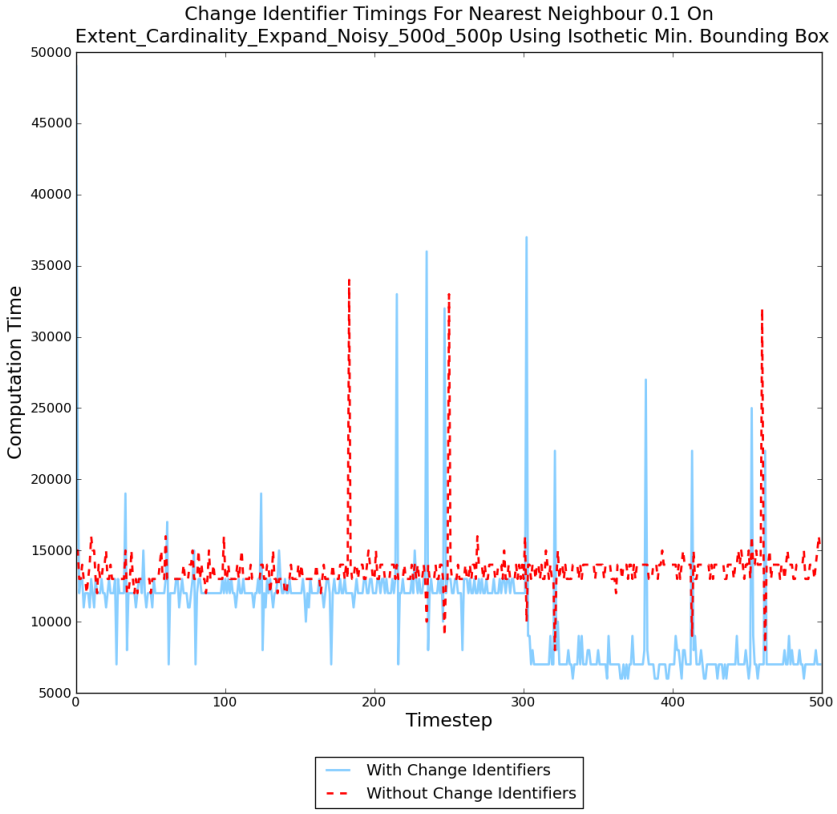


Figure 7.15 Time taken to compute at each timestep for NearestNeighbourDistVariance–0.1 on the Extent-Cardinality-Expand-Noisy dynamic dot pattern using the isothetic minimum bounding box footprint algorithm.

the experiment the mean proportional symmetric area difference was 9.18605×10^{-5} and the maximum was 0.04102977128896485; again we feel confident stating that this is well within the bounds of acceptability in terms of accurately tracking a footprint.

The NearestNeighbourDistVariance–0.1 identifier performs very well but by using the knowledge that we have about the nature of a specific dynamic dot pattern and the classes of other change identifiers we can define a set that reduces the symmetric area difference further without increasing the time taken greatly. For the final result of this chapter we define the user selected set UserSelected with the specification given in Listing 7.1. The identifier set has three identifiers: a measure of extent (DiameterSq), position (Centroid) and distribution (NearestNeighbourDistVariance) and a proportionate max allowed fails of 0.33. The threshold is set at 0.33 so that should any of the identifiers have their thresholds breached the footprint will be updated. The identifiers that have been chosen were selected to match the pattern which increases its extent by increasing the cardinality. The dynamic dot pattern will, therefore, directly affect the change identifier classes of distribution and extent. The position class identifier has been included to account for the noise shifting the centroid of the pattern.

The result of running UserSelected over the noisy extent dynamic dot pattern is shown in Fig. 7.20 plotted against two variants of NearestNeighbourDistVariance. The UserSelected has clearly increased the accuracy of the footprint tracking for a very small increase in time (the time taken axis has been made logarithmic to make the time

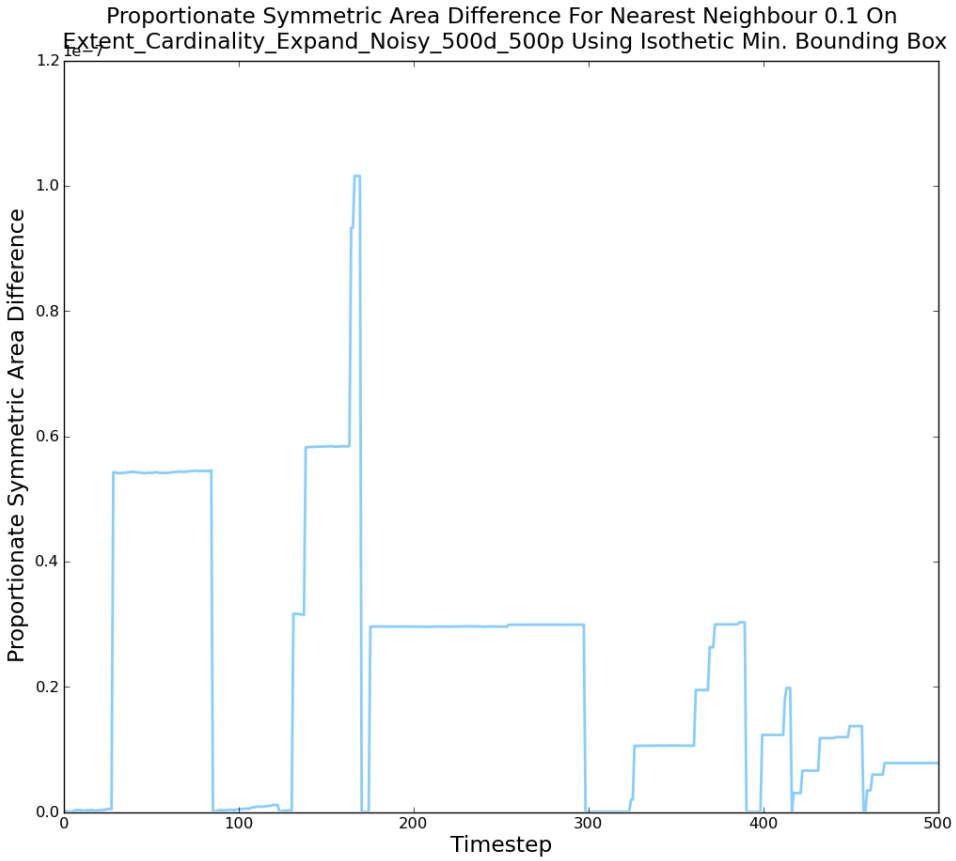


Figure 7.16 Proportionate symmetric area difference at each timestep for NearestNeighbourDistVariance–0.1 on the Extent-Cardinality-Expand-Noisy dynamic dot pattern using the isothetic minimum bounding box footprint algorithm.

distinctions clearer).

7.6 Summary

This chapter has shown that the change identifiers can decrease the number of updates needed to maintain a suitable footprint over a dynamic dot pattern. The graphs presented have demonstrated the reduction in time taken needed for computation per timestep, and they have also provided evidence that the error score for the stored footprint, when measured for similarity against the true footprint, is quantifiably low.

The chapter has also indicated that some change identifier sets perform measurably better than others and that, with some thought, it is possible to define change identifier sets that improve on the performance of other. The next chapter will look closer at the difficulties in change identifier selection.

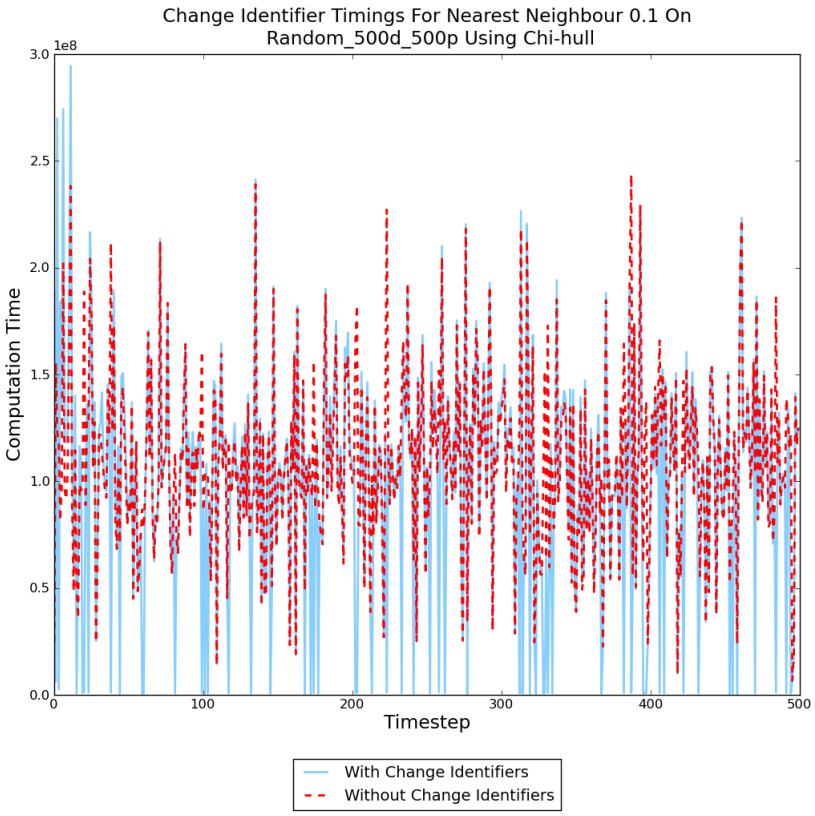


Figure 7.17 Time taken to compute at each timestep for NearestNeighbourDistVariance–0.1 on the Random dynamic dot pattern using the χ-hull footprint algorithm.

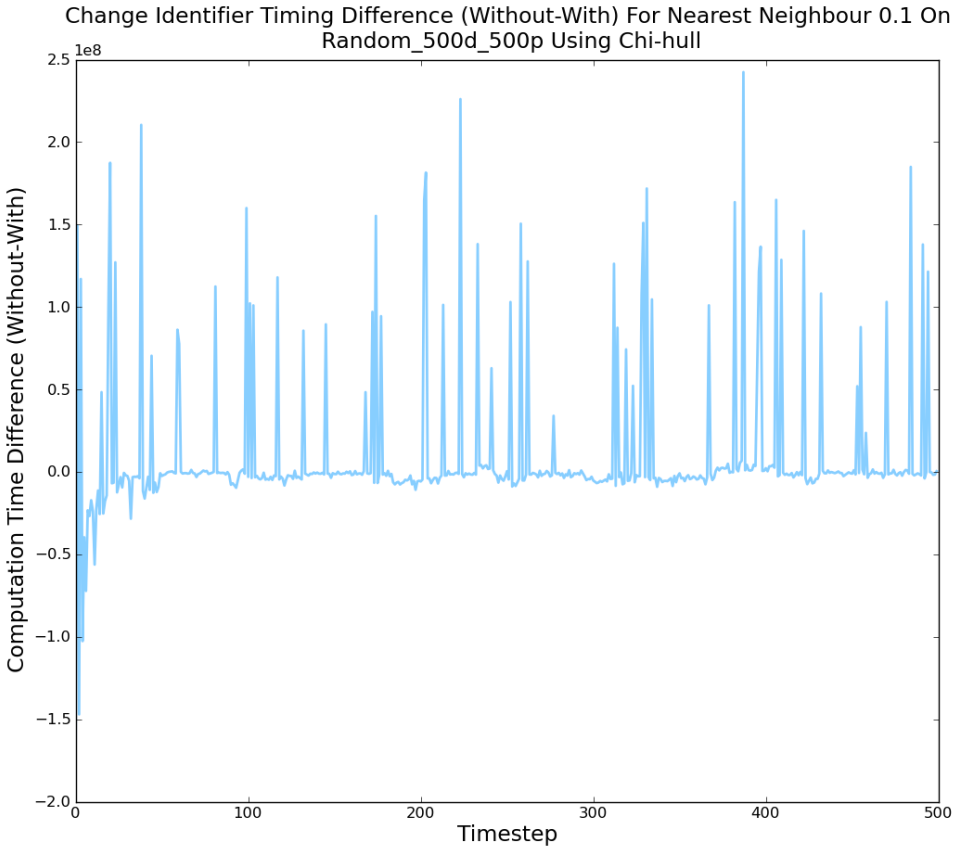


Figure 7.18 Difference in time taken to compute at each timestep for NearestNeighbourDistVariance–0.1 on the Random dynamic dot pattern using the χ-hull footprint algorithm.

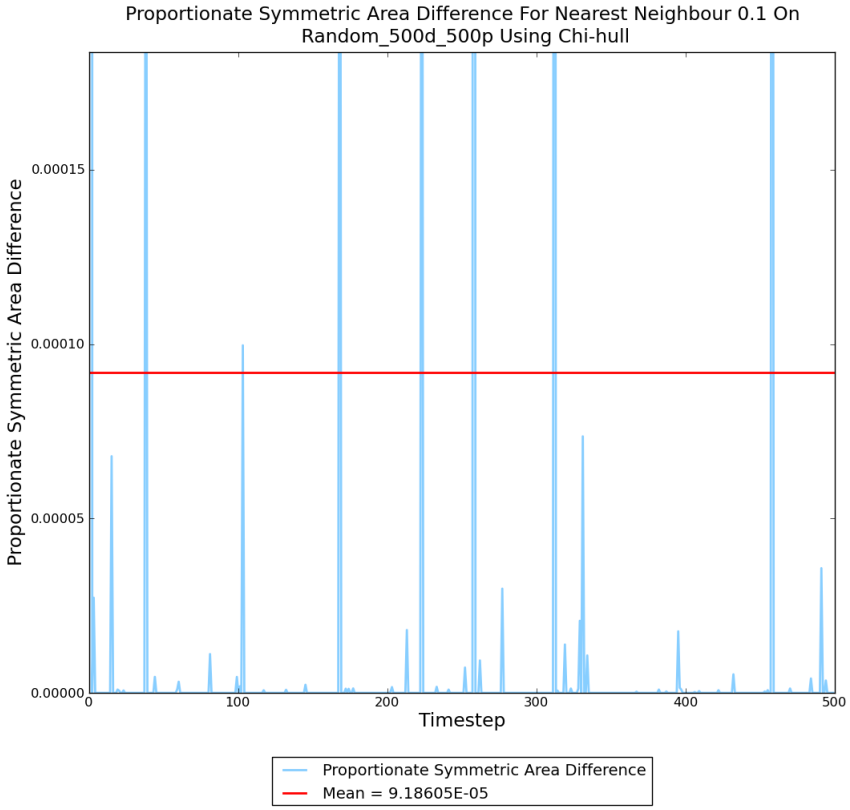


Figure 7.19 Proportionate symmetric area difference at each timestep for NearestNeighbourDistVariance–0.1 on the Random dynamic dot pattern using the χ-hull footprint algorithm.

```
<changeidentifierset name="UserSelected-0.33" ver="0.1">
  <description>User selected set</description>
  <maxFails proportionate="true">0.33</maxFails>
  <concurrent>false</concurrent>
  <changeidentifier>
    <classname>NearestNeighbourDistVariance</classname>
    <priority>0</priority>
    <threshold>0.1</threshold>
  </changeidentifier>
  <changeidentifier>
    <classname>DiameterSq</classname>
    <priority>0</priority>
    <threshold>0.1</threshold>
  </changeidentifier>
  <changeidentifier>
    <classname>Centroid</classname>
    <priority>0</priority>
    <threshold>0.1</threshold>
  </changeidentifier>
</changeidentifierset>
```

Listing 7.1 User Defined Change Identifier Set

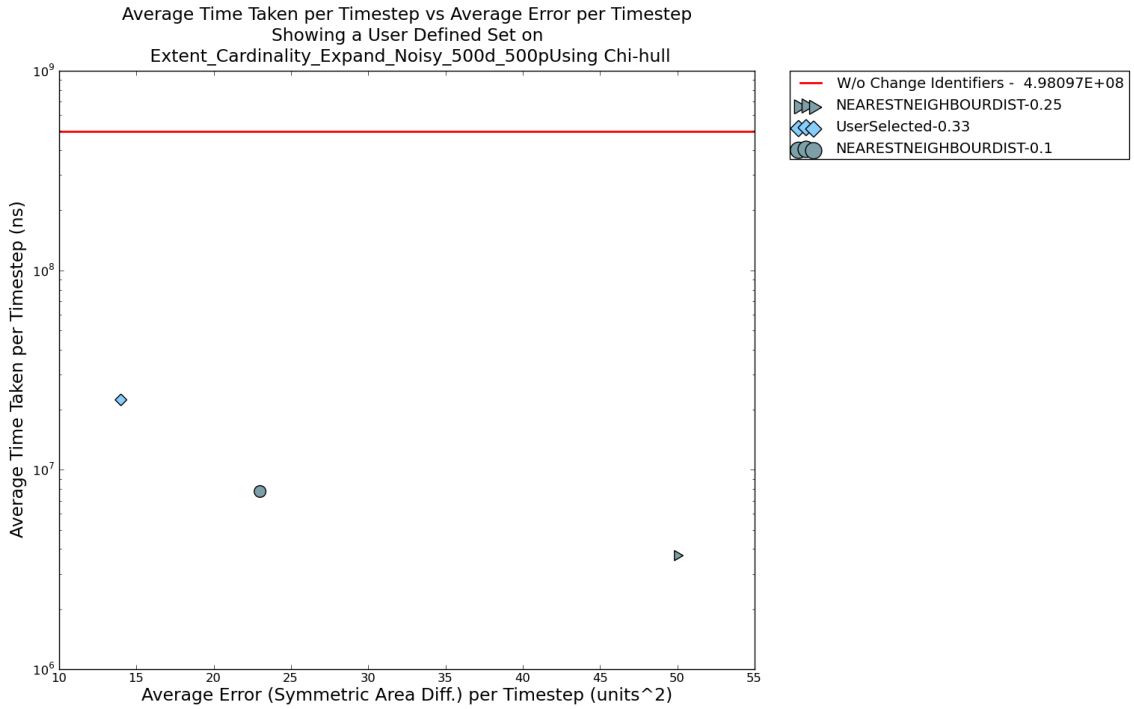


Figure 7.20 Time taken against error per timestep for NearestNeighbourDistVariance–0.1, NearestNeighbourDistVariance–0.25 and UserSelected–0.33 on the Extent-Cardinality-Expand-Noisy dynamic dot pattern using the χ-hull footprint algorithm.

8 Change Identifier Selection

The examination of the results of the change identifiers, individually and within sets, shows that careful selection of identifiers and their thresholds allows us to traverse the time against error trade-off. From this examination it can be seen that some sets perform better than others in both objectives. The difficulty that is presented to us is to identify whether there are some sets that always out-perform others[1], or, if not, is there a sensible method by which to select the appropriate set for any given application context?

To identify sets that always out-perform others is not a simple task. There are, given only the seventeen identifiers presented in this thesis, 131071 possible combinations[2] of identifiers (assuming no identifier can appear twice) to be tested against all dot patterns, preferably over more than one instance of each dynamic dot pattern type. Further complexity is added by the options in the construction of the set (for example, priorities for the identifiers, thresholds to use, maximum number of identifiers allowed to exceed their threshold, etc.). As the thresholds are real numbers the variability of the sets is infinite. Incrementally stepping through the sets using the framework would be a slow and unending task with no guarantee that the enumeration of the sets will find any conclusive evidence of any identifier outperforming any other. Fortunately there exist procedures with which to automate the checking and comparison of change identifier sets, such procedures are found within the field of optimisation. It should be noted that within this thesis we are not aiming to break new ground within the optimisation field. We simply wish to make use of its techniques to see if they can help with the selection of change identifier sets.

The graph of time taken against error can be seen to represent the space of all possible outcomes of the change identifier sets for a particular dynamic dot pattern (or averaged across many dynamic dot patterns). In Fig. 8.1 this space is represented by the shaded part of the graph. Spaces such as that depicted in Fig. 8.1 are called solution spaces; each solution in our solution space is a possible change identifier set. We can envision a second space for which the axes are the properties of the change identifier sets (e.g. theshold, concurrency, etc.) and, therefore, each point in this second space (or parameter space) represents a change identifier set. To provide clarity we assert that the parameter space is a multi-dimensional space with axes representing thresholds for each identifier, total threshold, maximum allowed identifiers to break their threshold as a proportion of the total number of identifiers (for brevity we will refer to this as the set's *proportionate threshold*) and any other parameters required to fully describe the set. Such a space is difficult to visualise but for our purpose it is enough to know that it exists. Each change

[1] Conversely if some always underperform

[2] $\sum_{k=1}^{17} \frac{17!}{k!(17-k)!}$

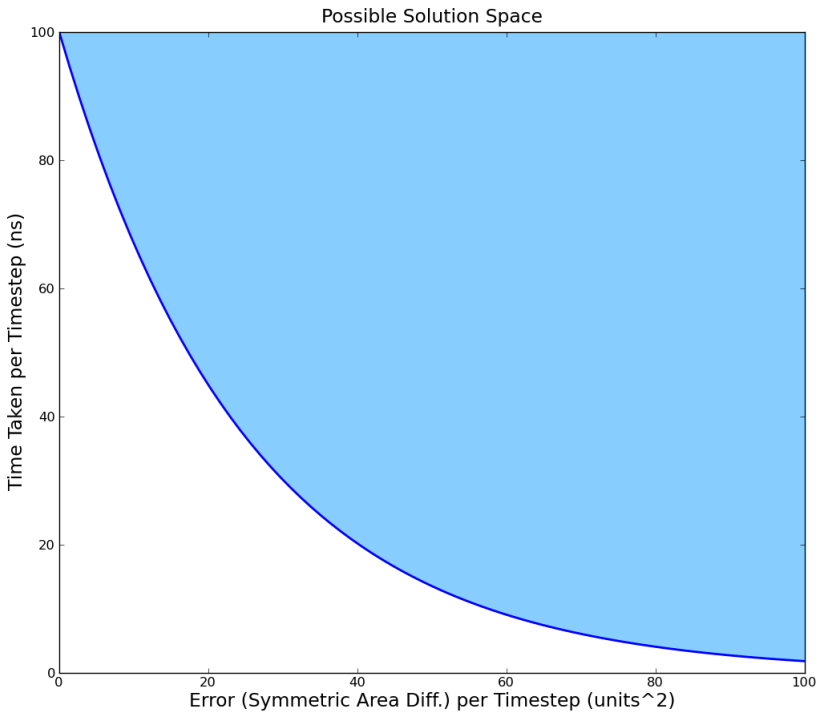


Figure 8.1 Graph showing time against error possible solution space.

identifier set in the solution space maps to a location in the parameter space. By defining the parameter and solution space we have specified an optimisation problem.

8.1 Optimisation

There are many possible optimisation techniques ranging from the stochastic to the enumerative. Given the large number of possible set combinations and the infinite variability provided by the real numbered thresholds an enumerative process suffers from much the same problem as iterating over the framework with an XML file for each set, in that it will be very slow. A stochastic approach, however, is more suited to multi-objective optimisation problems as it can move faster and is less prone to falling into what are called local optima, in which the solutions surrounding the current best solution found perform worse but there exist better solutions in other parts of the possible change identifier set space. We will be using a stochastic approach based on ideas of mimicking the organic evolutionary process. As such the best, often called the fittest, solutions survive to propagate through various generations of the search [31]. The solutions will converge towards the optimum after a, often large, number of generations. Importantly there is no way of knowing if the converged upon solutions are truly optimal. However we are interested in knowing if some identifiers out perform others, not if we have reached the true optima and so do not concern ourselves too much with convergence.

Methods for solving optimisation problems based on evolutionary theory are called ‘Evo-

lutionary Algorithms' (EAs); Deb [16] gives a comprehensive overview of their history and construction and will act as the background for much of the description given below. EAs can be reduced to four main actions: Selection, perturbation, evaluation and update. The selection stage takes from the pool (*population*) of solutions one or more candidates. These canditates are then perturbed in some fashion (e.g., *mutation* or *crossover* – we will discuss both shortly). The perturbed candidates are evaluated against some test of appropriateness for solving the application problem (*fitness function*) and, if their performance is better than existing solutions in the population, then the population pool is updated. This four stage process is iterated until a desired value is achieved for the fitness function or a pre-determined number of iterations (*generations*) is reached.

The fitness function for the optimisation of change identifiers is given by running the change identifier framework, using the candidate set proposed by the EA, over one or more dot patterns and taking measurements of its error and time taken as was performed in the results chapter. The objectives, as stated, are the minimisation of error and the time taken. These two objectives are not unrelated, we can decrease error by increasing the number of updates which then increases the total time taken. When dealing with connected objectives we may wish to find a function with which to combine them so that single objective optimisation techniques may be applied. However, this approach leads to possibly unfair weighting of one objective over another and the conceptual error of making an 'apples to oranges' comparison. Instead we can treat the objectives as both equally desirable and use Multi-Objective Optimisation Problem (MOOP) techniques. Given these interacting, but distinct, criteria we look not for the best set but the Pareto optimal (sometimes called Pareto efficient) sets. If a solution can not improve performance in one objective without decreasing performance in another then it can be said to be Pareto optimal. Within the MOOP field the terminology of dominance is used to describe the Pareto optimality of a solution. The following statements use Fig. 8.2 as an example of dominance relations in which the aim is for both objectives to be minimised.

- Strong Dominance: One solution outperforms another on all objectives. So in Fig. 8.2 $\forall x(x \in O \wedge a(x) < b(x))$ in which O is the set of all objectives.
- Weak Dominance: One solution out performs another on at least one objective and is no worse on any other. $\exists x(x \in O \wedge d(x) < b(x) \wedge \forall x\ d(x) \leq b(x))$.
- Mutual Non-Dominance: One solution performs better than another on at least one objective and worse on at least one objective. $\exists x(x \in O \wedge a(x) < d(x)) \wedge \exists y((y \in O \wedge a(y) > d(y))$.

To fully explain the figure: a, d and e are mutually non-dominating, b is dominated by a and weakly dominated by d, and c is dominated by d and e.

There are many different types of EA but, as the construction of the change identifier sets contain real numbered values (the thresholds), we can not use the majority of Genetic Algorithm variants; they require solution specifications that have fixed size sets of possible

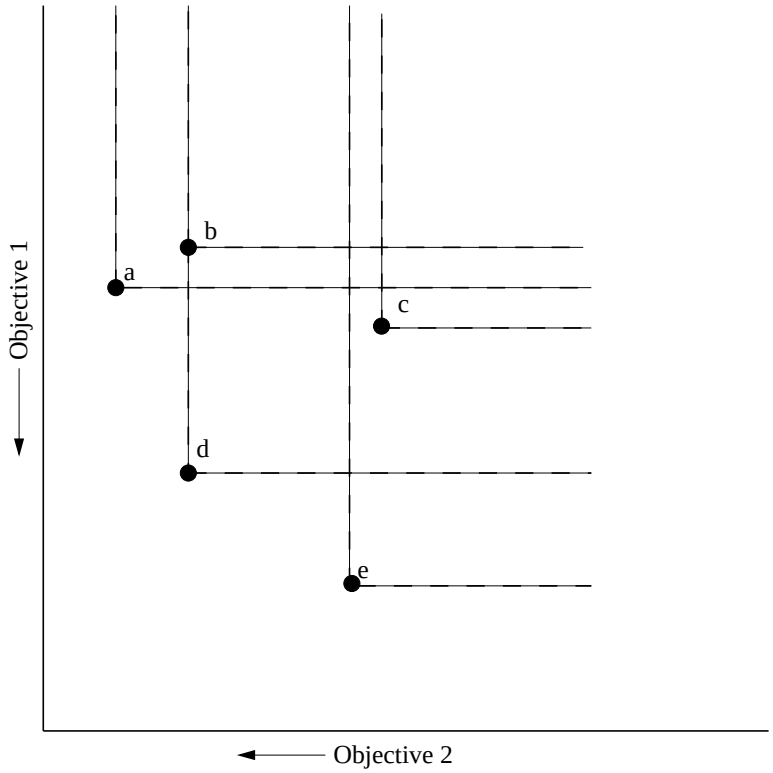


Figure 8.2 Graph showing dominance relations.

values[3]. A similar method type[4] is the Evolution Strategy (ES) which does not have an objection to the use of real number values. The particular ES used in this thesis is a $(\mu + \lambda)$-ES, in which μ represents the size of the population pool and λ represents the number of the permutations (or mutations) every generation. We need to encode our change identifier sets as a vector of parameters so that the permutations can be applied, Fig. 8.3 shows how this encoding is formed.

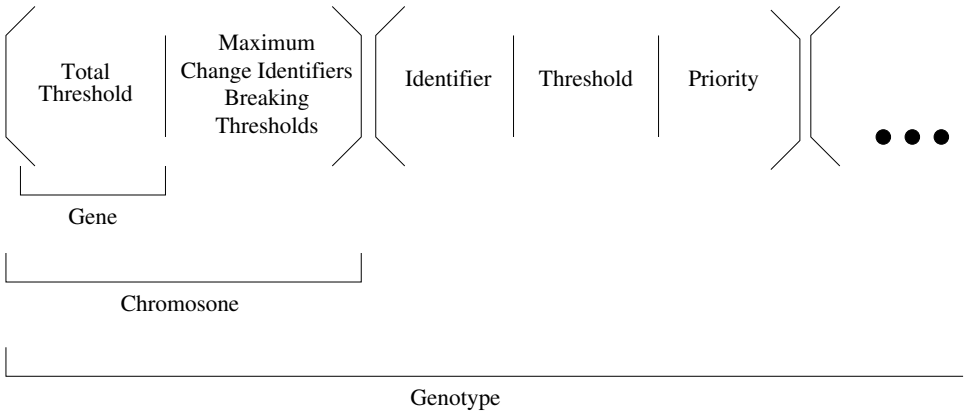


Figure 8.3 Encoding of the Change Identifier Set.

Note that we have not encoded all the possible parameters. We have removed the option of concurrence from the set as a whole and the options for multipliers or forcing an update when a threshold is breached on the individual identifiers. Without this simplification an identifier may end up with a multiplier so low that it never causes a change, or so high that

[3]Real valued GAs now exist, making the distinction between a GA and an ES somewhat blurry.
[4]Produced in isolation but at almost the same time

it repeatedly forces one; riding through to the final generation while never contributing to the effectiveness of its set, or hampering its set by forcing unecessary updates.

The terminology used is comes from the field of genetic science and serves to re-inforce the relationship between the natural process and the artificial [31]. The structure, or genotype, is split into chromosomes; each representing some part of the set. At chromosome locus (position) 1 is the chromosome representing the total set details. The chromosomes are made up of genes, each representing a single piece of information, so the gene in chromosome 1 at gene locus 1 represents the total threshold. The value assigned to any gene alters the expression of that gene, a gene with a specific value is therefore called an allele.

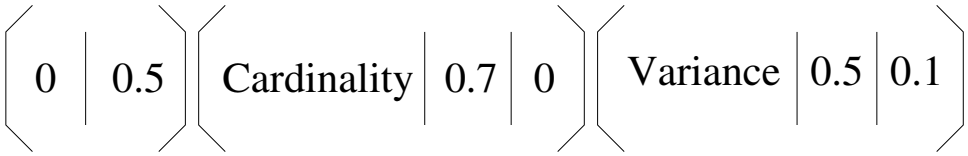


Figure 8.4 Example encoding of the change identifier set.

The best μ candidates of the union of the initial set and the permutations pass through to the next generation. The mutation is performed by addition of a value with a normal distribution to every real value of the solution vector. In our case the encoding is of mixed type and we do not wish to apply the mutation to the change identifier, randomly changing the identifier would result in large jumps around the solution space and may hinder any progress made by the real value mutation. Instead we make sure that the initial population features several instances of each identifier across the sets and apply a crossover operator after the mutation step. Crossover mimics the propagation of genetic material via procreation. A breeding pool of the best performing solutions is created in which each solution is randomly assigned a partner with which it produces two offspring by swapping a randomly selected chromosome. If the offspring outperform their parents they replace them in the next generation. Within our selection process, after the mutation, the set of non-dominated solutions are identified and removed from the candidates (the union of the initial set and their mutations) and added the set of solutions for the crossover process. This is repeated on the remaining candidates until μ solutions are in the next generation set. This process identifies the *Pareto* layers of the generation [56], peeling them off one by one[5]. For the crossover operator we simply use the first layer of the non-dominated solutions as our breeding pool.

8.2 Results of the ES

There were several different optimisation experiments that were run for this chapter that demonstrate different concerns to be considered when looking at change identifier selection.

The first experiment was run with a μ (population) of 15 and a λ (perturbations) of 30 for fifty generations on a single dynamic dot pattern (the noisy change in extent by cardinality

[5]The author invites the reader to draw their own analogies as the onion is overdone; he suggests rock strata as a possibility.

pattern used in the previous chapter) and using the χ-hull algorithm.

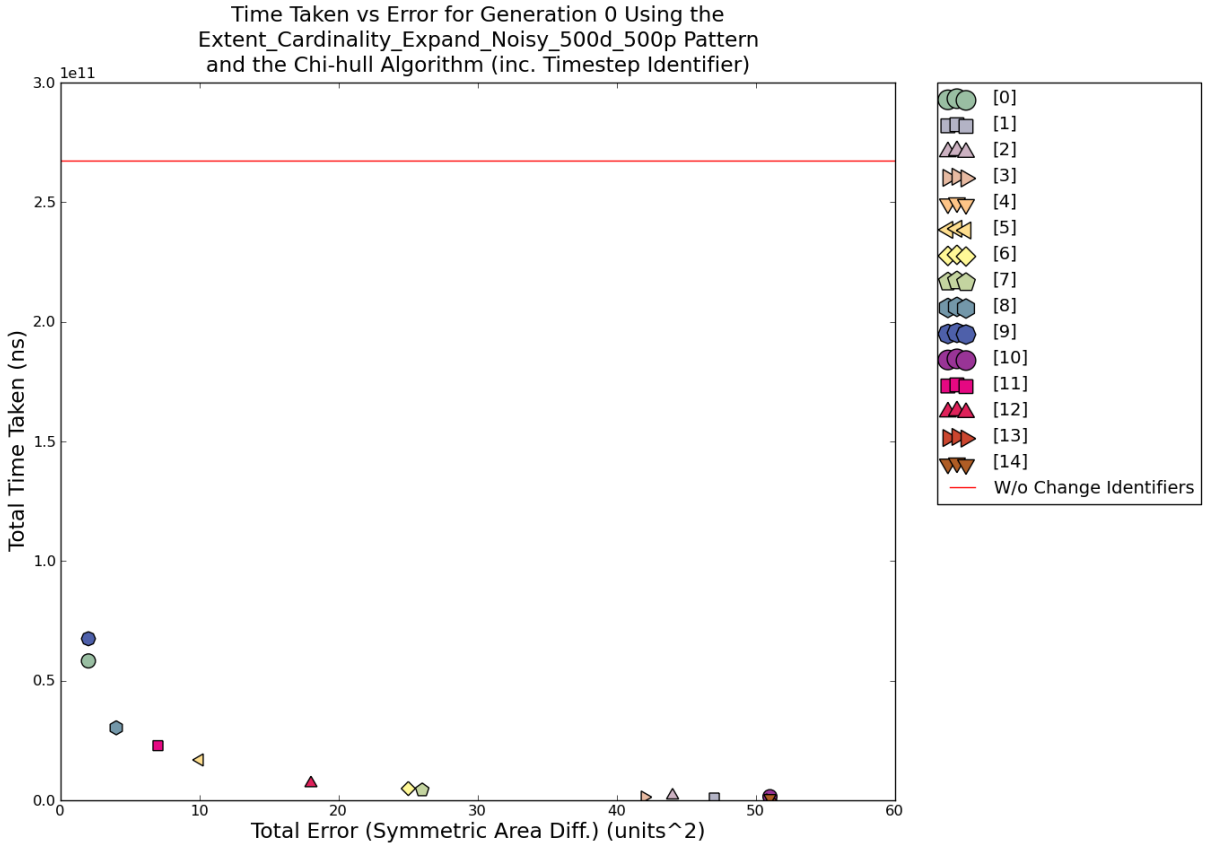


Figure 8.5 Initial randomly created population of change identifier sets for an ES running on the Extent-Cardinality-Expand-Noisy dynamic dot pattern using the χ-hull footprint algorithm (incl. TimeStepCount).

Fig. 8.5 shows the initial population for this experiment. This population performs surprisingly well given its unconstrained random generation, with every identifier set well below the line representing a **without** run (T_{NCI} from Chapter 5) and with small error values, some of the identifier sets even performing better than the user selected set from the previous chapter. The numbers are Serial Numbers (SN) given to every candidate when it is created so that unique identifier sets can be tracked through the run of the ES. Each SN maps to a listing of the genotype that describe it. For example, the identifier set with SN [9] has the genotype:

```
[0.0,0.20435381]
[OLSGradient,0.3493427,0.7897132]
[Centroid,0.17240429,0.41104406]
[SampleCoeffCorr,0.21067989,0.82546014]
```

Using the above description of the encoding this translates to identifier set [9] having a total threshold of just over 20% and containing three identifiers. OLSGradient, Centroid and SampleCoeffCorr have the thresholds $\approx$ 35%, $\approx$ 17% and $\approx$ 21% respectively. The three identifiers are run in the sequence: Centroid, OLSGradient and finally

SampleCoeffCorr. The final generation of this optimisation result gives Fig. 8.6 in which there has been improvement made but it has been, expectedly given the initial population, slight. Despite having only a minor improvement the ES has presented some interesting phenomena:

- Identifier set [2492] appears to have a total symmetric area difference of 0 and is still below the time taken to compute a **without** run by 13.5 seconds[6]. The total time to run **without** is 267.6 seconds[7] and it is conceivable that during this time the computer the experiment was running on tied up the processor for another task. From the result of [2492] we can conclude that for a **with** run to be considered as performing faster than a **without** it must run in a significantly faster time[8].

- Identifier sets 0.08 [1021] and 578 have identical symmetric area difference and near identical times (0.09 seconds[9] difference) despite having different genotypes:

```
[578] = [0.09240082,0.0]
[Cardinality,1.8340104,-0.8159473]
[TimeStepCount,0.5519325,0.9534129]
[Centroid,0.5590967,0.9993394]
[DensityIdentifier,0.08898324,0.87055963]

[1021] = [0.09240082,0.0]
[Cardinality,2.5320697,-0.4189783]
[DensityIdentifier,1.8247916,1.6922433]
```

  The genotypes do have two identifiers in common and only require one of their identifiers to breach its threshold to force an update as they both use the proportionate threshold to indicate when a footprint should be updated. They also both have the same proportionate threshold value of 0.09240082, which accounts for less than the proportion given by any identifier threhsold breach (0.25 for [578] and 0.5 for [1021]). The low complexity of the not-shared identifiers may account for the similar times and the commonality within the genotype means that they have a strong chance of causing updates at the same phases – resulting in identical symmetric area differences.

- Seven of the identifiers contain the TimeStepCount identifier and all of them contain the Cardinality identifier. The Cardinality identifier having a strong representation is what would be expected from a dynamic dot pattern that changes primarily via its cardinality. The TimeStepCount identifier, however, is an adaptive identifier to the pattern; any 'gaps' in a set where the it fails to recognise change in a dynamic dot pattern for a range of phases can be filled by a TimeStepCount identifier with a threshold set to force an update at the phase(s) during this range.

[6] 13545555000 nanoseconds
[7] $2.67561768 \times 10^{11}$ nanoseconds
[8] 13.5 seconds is roughly only 5% of the total time
[9] 85668000 nanoseconds

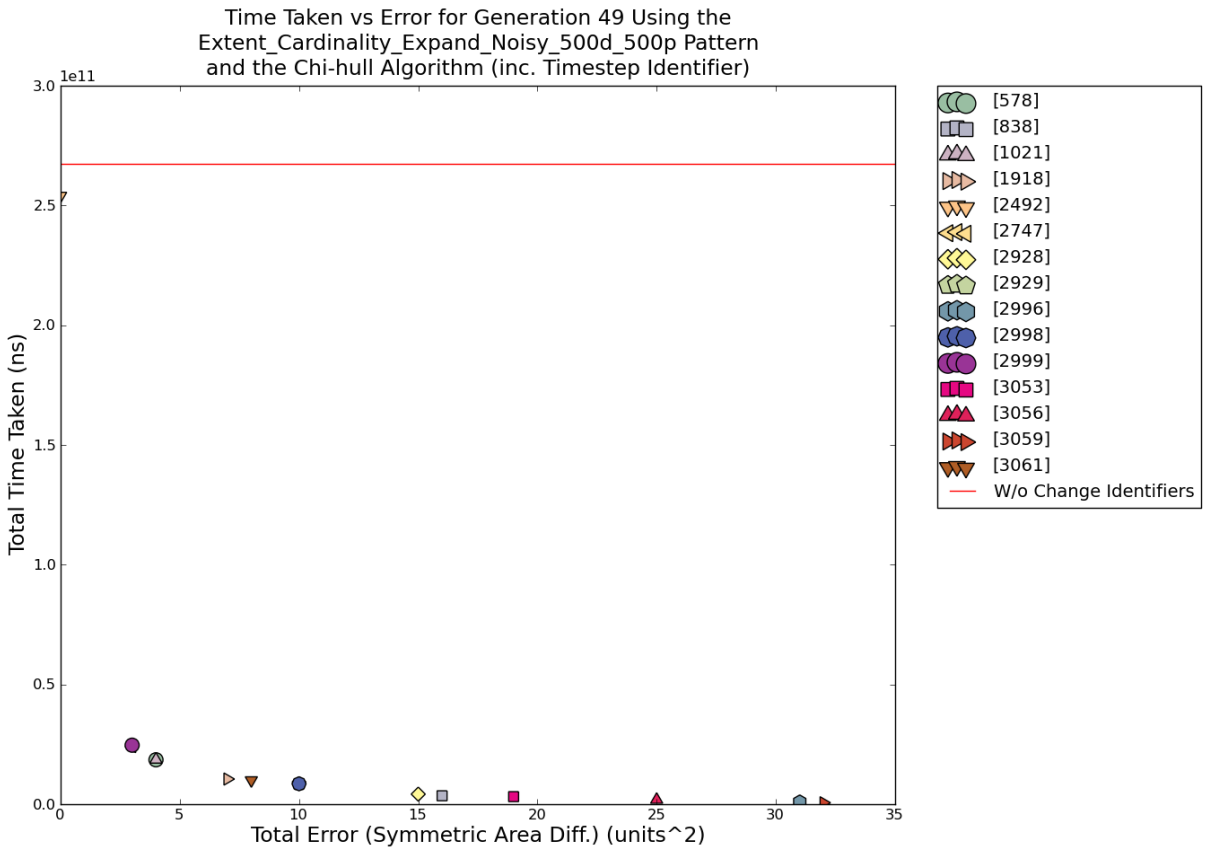


Figure 8.6 Last generation for an ES running on the Extent-Cardinality-Expand-Noisy dynamic dot pattern using the χ-hull footprint algorithm (incl. TimeStepCount).

If the TimeStepCount identifier is removed from the possible identifier sets then the results a different collection of identifiers come to the fore. In Fig. 8.7 all but three of the sets have the VarianceIdentifier, and all but one contain at least one identifier based on a descriptor that either measures extent directly or correlates with it (Kurtosis and Density). The exception is set [3059] which uses the OLSGradient. That the extent measures are so prevalent is encouraging, given that it is in extent that the dynamic pattern was constructed to change. The identifiers [2236] and [3074] both have a total error of 1, with [2236] running faster (by 9.3 seconds[10]) than [3074]. In comparison to Fig. 8.6 both run slower than [578], [1021] and [2999] but have less total error. This indicates that the extent measures can identify the change of the pattern better than the mixed TimeStepCount and PatternDotsCount but do so with a slight time cost.

As an observation on the shape of optimisation graphs shown for the extent pattern, it is noted that the region of the trade-off curves in Fig. 8.6 and Fig. 8.7 where sets [2999], [1021], [578], [2236], [3074] and [2319] are found is called the *knee* of the curve. The knee is of of particular interest in our experimentation because if it is shallow then decreases in error along it do not cause large increases in time and vice-versa; the more pronounced the knee the more drastic the decline in one objective for improvement in the other. Ideally the identifier sets that the ES produces will be as close to the apex of the knee as possible,

[10] $3.6426894 \times 10^{10} - 3.5494388 \times 10^{10}$ nanoseconds.

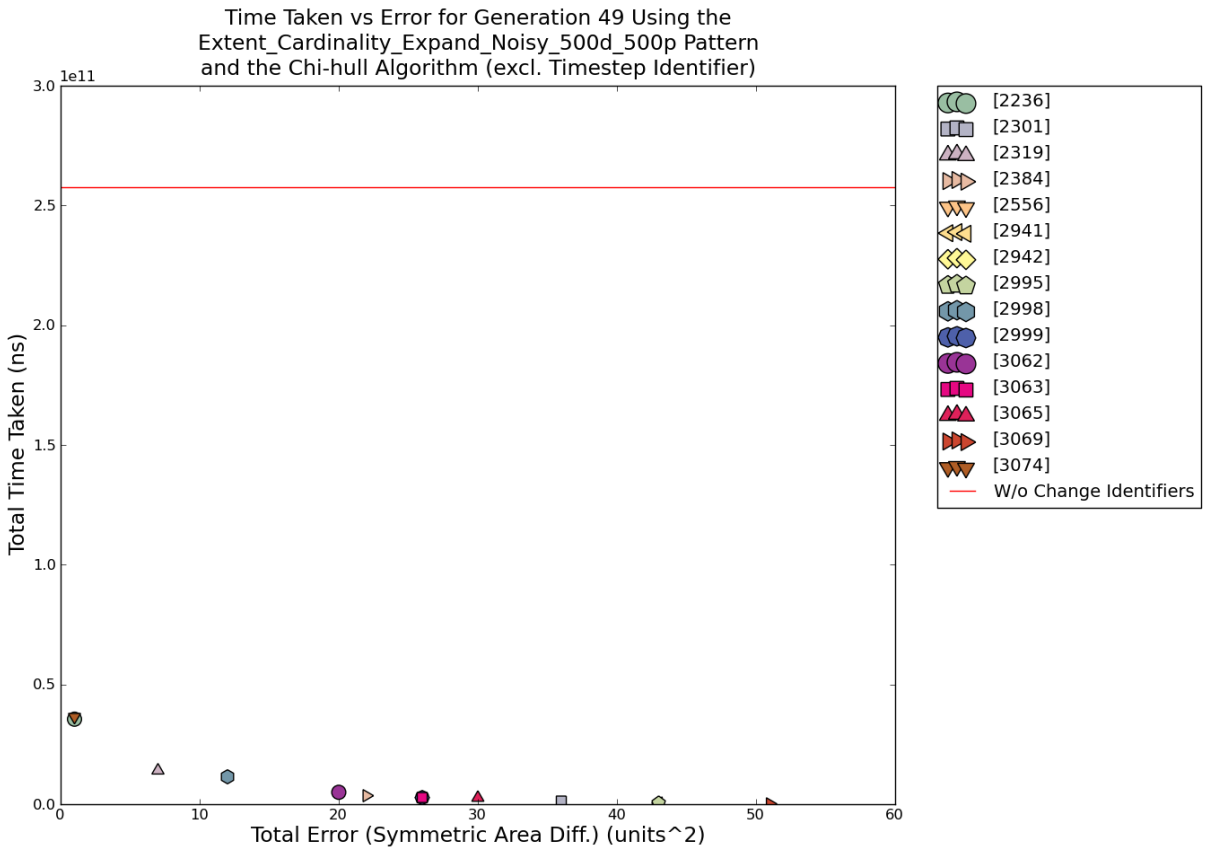


Figure 8.7 Last generation for an ES running on the Extent-Cardinality-Expand-Noisy dynamic dot pattern using the χ-hull footprint algorithm (excl. TimeStepCount).

as these represent an optimum balanced trade-off point.

The previous optimisation experiments are running over a single pattern, hence the sets they produce are over-fitted to the data. Over-fitting occurs when a solution only works well on the data used to generate it, and the sets given above are all specific to the extent change by cardinality pattern. By running the ES so that the fitness of the solution is averaged across the dynamic dot pattern types used in the results chapter the problem of over-fitting can be lessened. Fig. 8.8 shows the initial population for a $(\mu + \lambda)$-ES with μ of fifteen and a λ of thirty using all of the dot patterns used in the results chapter in its fitness function and the χ-hull algorithm. Fig. 8.9 shows the first generation produced by the ES and Fig. 8.10 the twenty-fourth. Running over all the dot patterns greatly increases the time taken to run the fitness function but by generation 24 convergance is already starting to occur. The collection of identifiers [1371], [941], [304], [1440], [881] and [1512] all use different thresholded variations of the same two identifiers: DiameterSq and Density. It must be made clear that this does not mean that these two identifiers are better than the others but that on the range of patterns tested they were the most consistent at reducing error and time taken.

The final graphs shown in this chapter are the average time taken per timestep (Fig. 8.11)

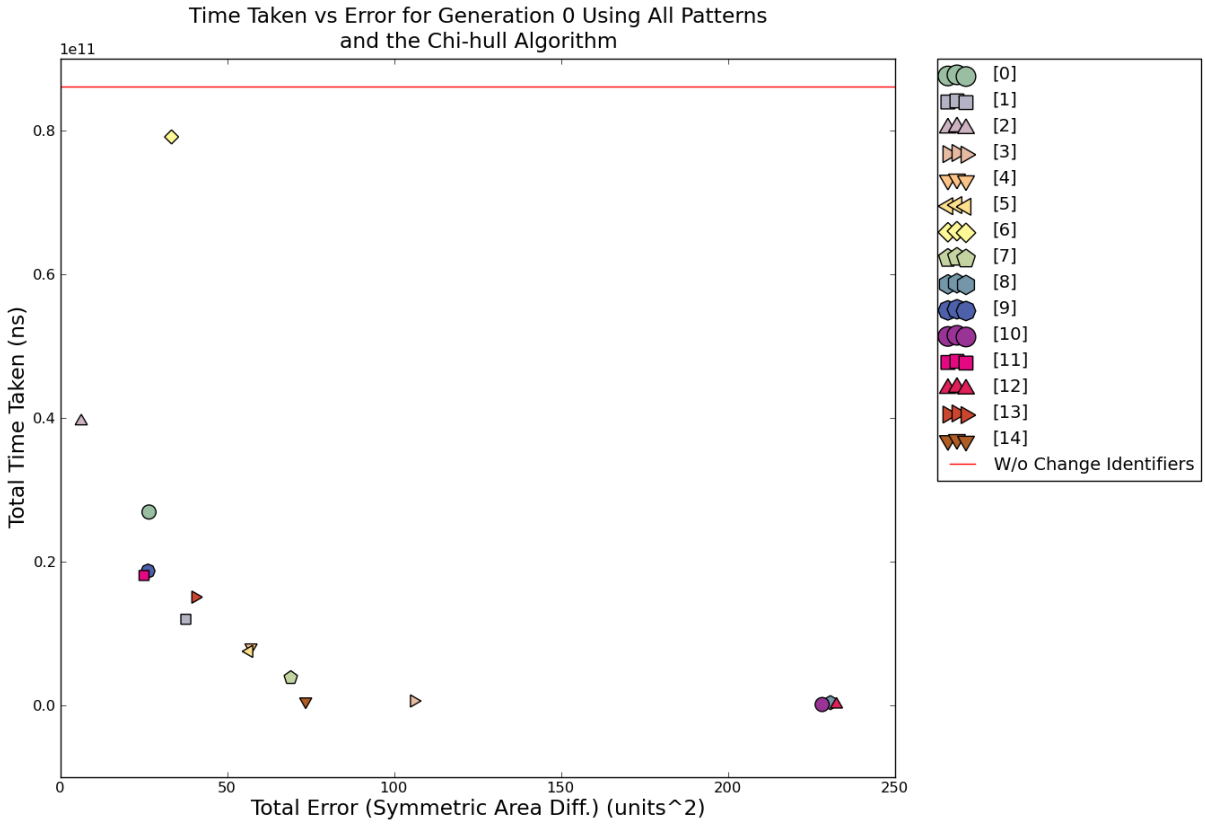


Figure 8.8 Initial randomly created population of change identifier sets for an ES running on all dynamic dot patterns using the χ-hull footprint algorithm

and the average proportionate error per timestep (Fig. 8.12[11]) on each dynamic dot pattern type for the identifier sets: UserSelected, NearestNeighbourVariance-0.1 and [304]. These represent the set selected via human intuition, a set selected by enumeration and a set selected by a stochastic optimisation process respectively. The time taken plot favours the optimised identifier but the user selected is often a close second. However it is the NearestNeighbourVariance-0.1 set which performs best on the error plot. Arguably the most important result demonstrated by this graph is that the UserSelected set reduces the error further than both [304] and NearestNeighbourVariance-0.1 on the noisy change in extent by cardinality pattern. This is important because [304] and NearestNeighbourVariance-0.1 were chosen as sets that perform well in general, while the user selected set was created specifically for that pattern.

8.3 Summary

This chapter has shown that optimisation techniques can be used to produce identifier sets that work with a reasonable degree of accuracy across a range of dynamic dot pattern

[11] The average proportionate error plot does not have values for orbit or running as the average symmetric area difference was 0, the *chi*-hull was unable to produce footprints with an area on these patterns because of their often collinear distributions.

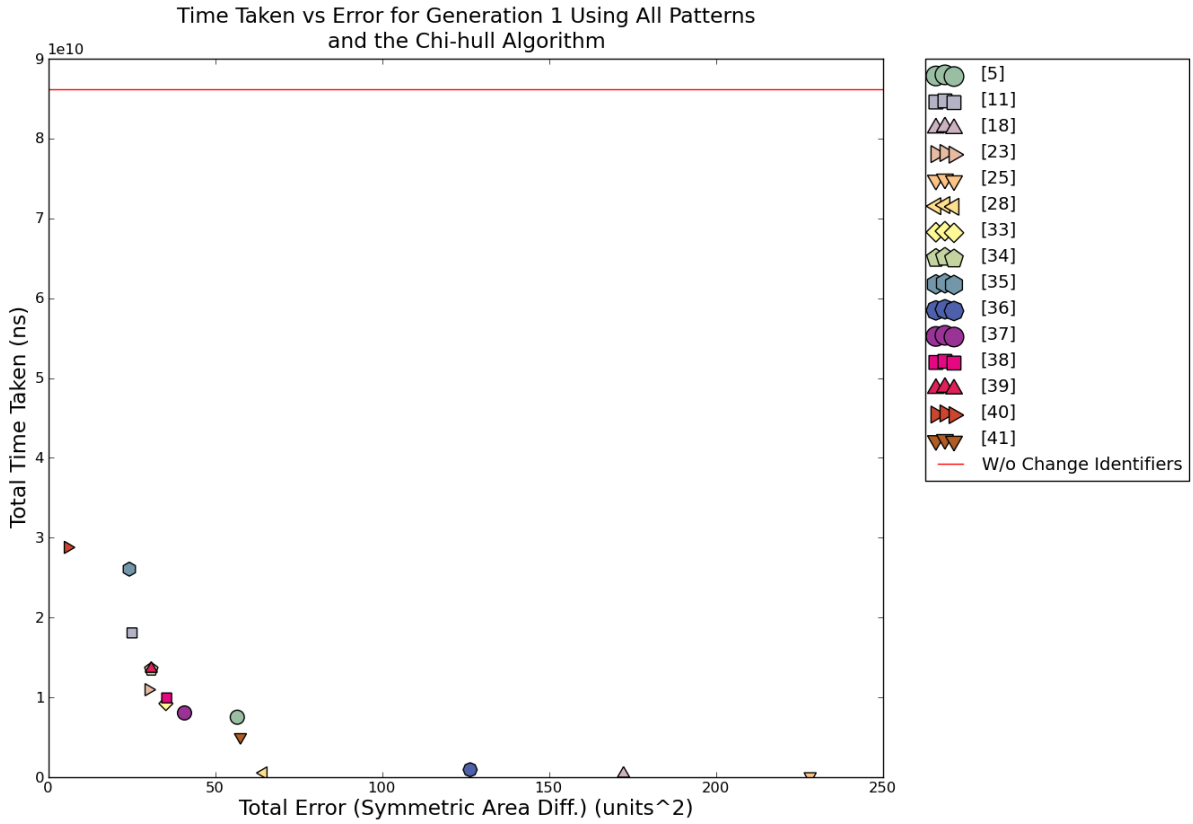


Figure 8.9 First generation for an ES running on all dynamic dot patterns using the χ-hull footprint algorithm

types. It is likely that this can be improved by the addition of more dynamic dot pattern examples. It has also provided further evidence to the statement given in the Chapter 7 that effective change identifier sets can be created by a user if they have knowledge about some of the descriptor classes that a dynamic dot pattern will change in.

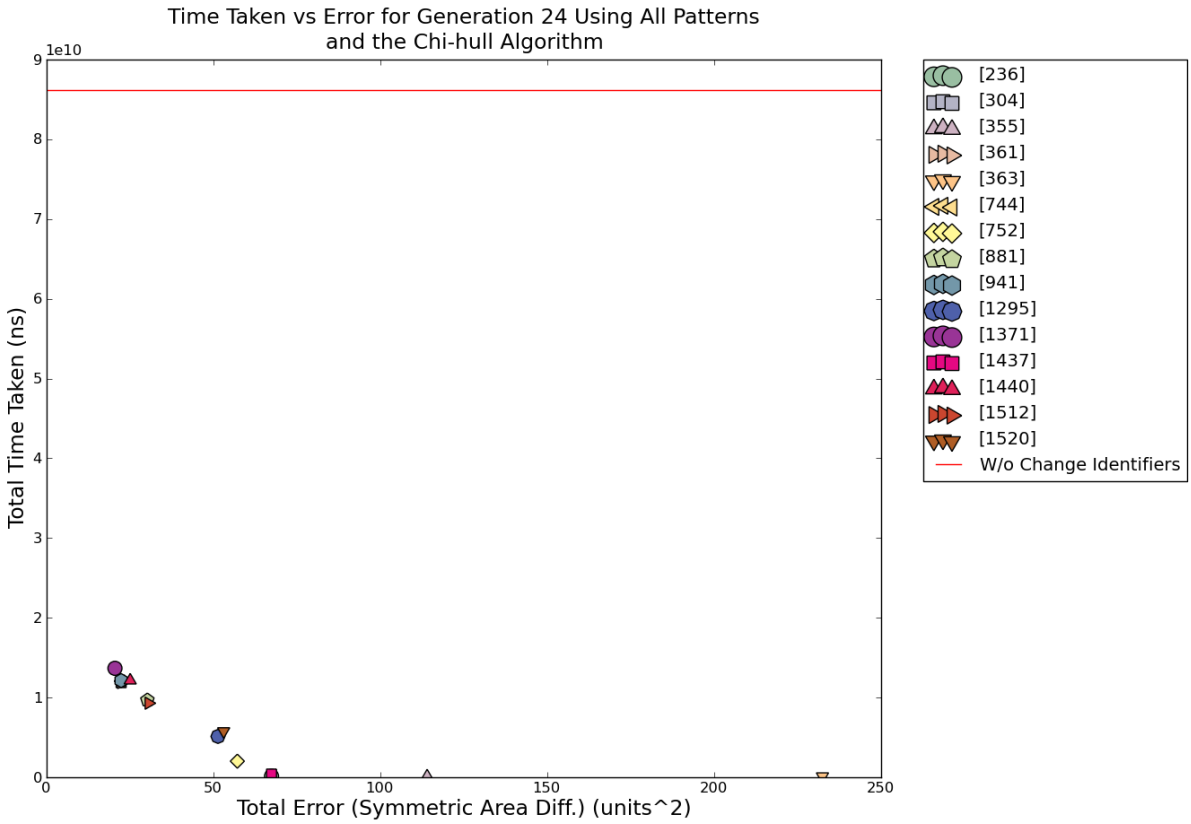


Figure 8.10 Twenty-fourth generation for an ES running on all dynamic dot patterns using the χ-hull footprint algorithm

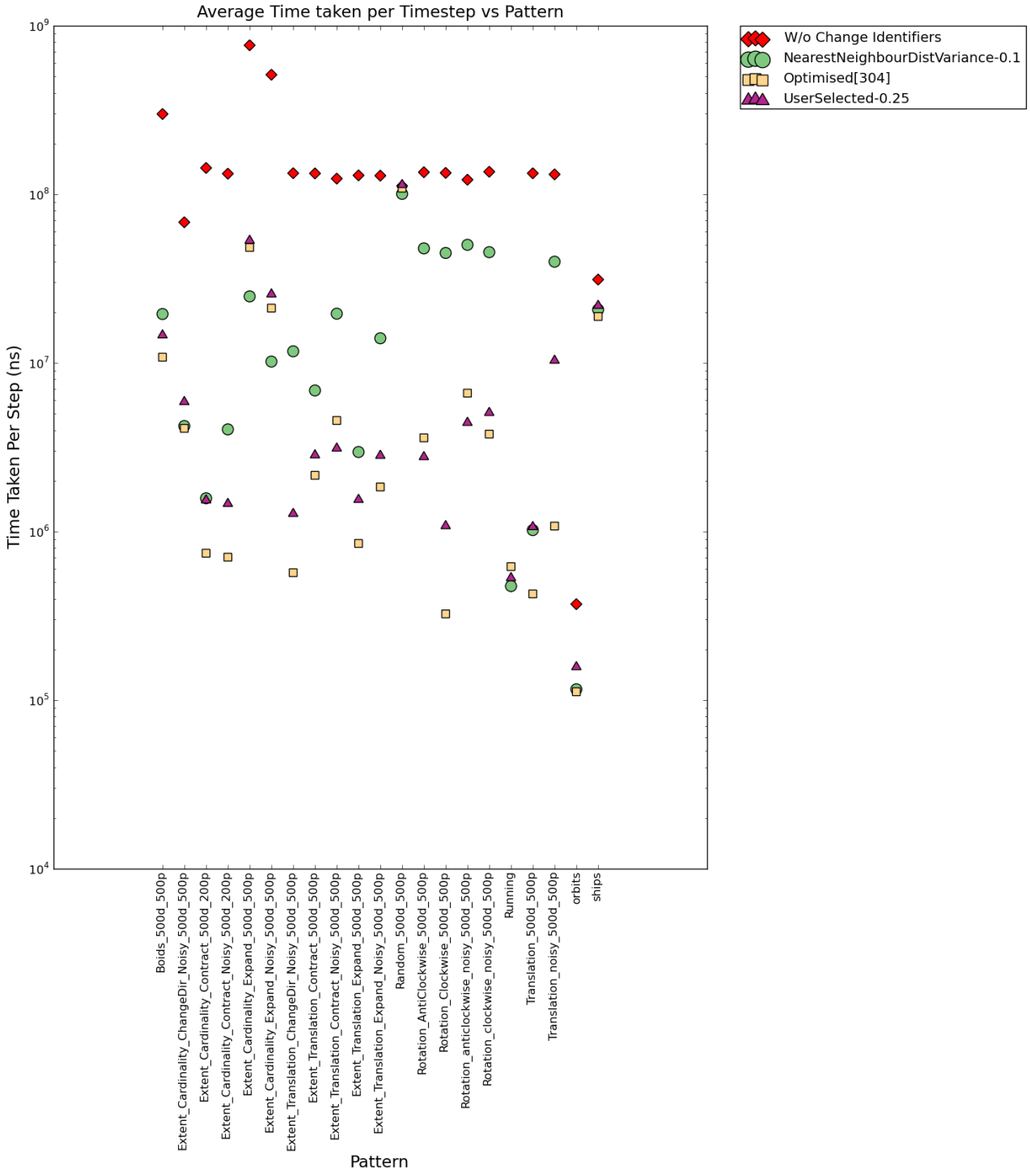


Figure 8.11 Average time taken per timestep for the UserSelected, NearestNeighbourVariance-0.1 and [304] change identifier sets on each dot pattern type.

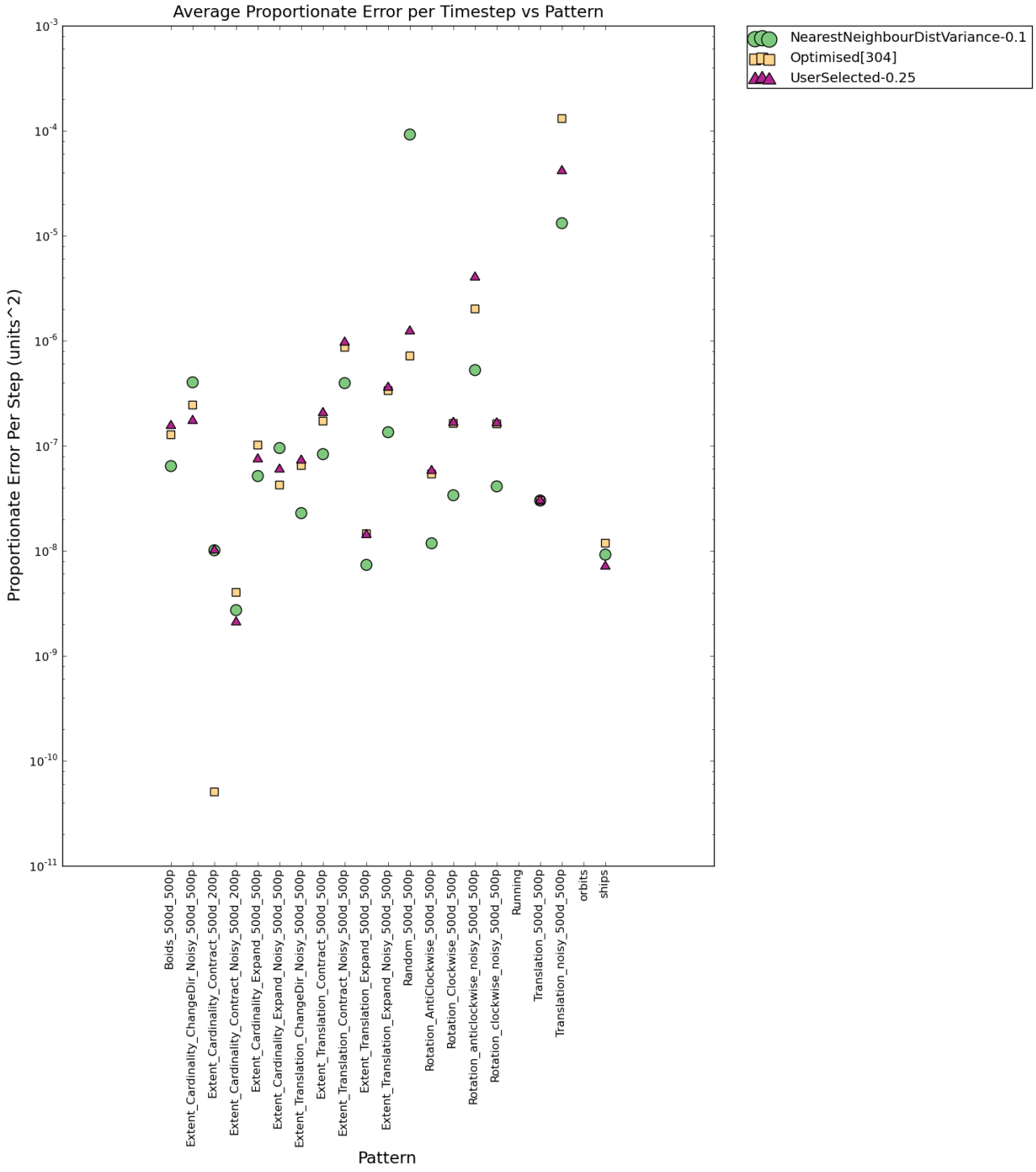


Figure 8.12 Average proportionate error per timestep for the UserSelected, NearestNeighbourVariance-0.1 and [304] change identifier sets on each dot pattern type.

9 Conclusions

The goal of this thesis has been to show that the use of change identifiers will reduce the time taken to maintain a footprint over a dynamic dot pattern while introducing an acceptable level of error.

The inquiry began with an investigation of the dot patterns. Our goal was, not merely to provide background to the change identifiers, but to see if measurements on the dot patterns could provide useful information in its own right. This exploration of the dot patterns led to the identification and analysis of the dot pattern descriptors. There is a wealth of information present in the individual patterns and the descriptors are measures of this data. It was found that, not only, could the descriptors provide an stable base for the change identifiers but that they may be able give a classification structure for the dot patterns. A preliminary examination of how this classification might be constructed is demonstrated in Chapter 10.

Footprints have a large scope of operation; appearing in different forms across a range of fields. Before looking at the change identifiers this thesis devoted a chapter (Chapter 4) to the investigation of the types of footprint that are commonly produced by the footprint algorithms in the literature and proposed a classification (as an extension of the work performed in [21]) based on this investigation. The chapter also looked at how the footprint type may be affected as it is updated over a dynamic dot pattern; further discussion on which can be found within the future work chapter.

Having discussed the underlying aspects of the proposed problem the thesis examined the change identifiers in greater detail. Chapter 5 presented the change identifiers used within this thesis and proposed a method with which to combine them into sets measuring mutliple different types of change. It also introduced an assessment approach based on comparing the stored footprint the change identifier set presented at any particular dot pattern phase with the 'true' footprint, i.e. the footprint that would have been created had the algorithm been run at this timestep, for the same phase.

With the change identifiers defined and an assessment method in place the experimentation could be performed. The results of these were shown in Chapter 7, which also detailed the range of dynamic dot patterns and algorithms that were used to provide a fair appraisal ofe change identifier sets' worth. Chapter 7 showed that it was possible to reduce (greatly in some instances) the number of footprint updates while maintaining a symmetric area difference that was low proportional to the area of the 'true' footprint. The chapter also demonstrated that some identifier sets can out-perform others for specific patterns and that it is possible to create sets to do so using knowledge about the dynamic dot pattern's

nature. Finally the results indicated that some identifiers may be generally more applicable than others.

Choosing the appropriate set for any given application is the main difficulty that may arise when using them. For some applications it may be easy to know in advance what type of change is most likely to occur and therefore which identifiers to use, however there are some applications in which the change can occur in different and unpredictable ways. In such erratic cases we need to find a set of identifiers which can identify a mixed range of change types while still making time savings. The chapter on change identifier set selection Chapter 8 undertook this search using Multi-Objective Optimisation techniques which converged on some identifiers that were generally applicable to the test data used. It was noted, however, that this did not indicate that these identifier were superior to others, but that they are useful in many situations.

This thesis has presented the concept of change identifiers and shown that they can be used to reduce the number of footprint updates required to have a suitable representation of a dynamic dot pattern with a user controllable error trade-off. It has provided an initial set of change identifiers and a framework in which they can be used. Further to the use of change identifiers as a way of reducing the number of updates other uses they might have were explored. This exploration led to the conclusion that the information supplied by the change identifiers could well be useful in its own right, perhaps even by-passing the need to produce a footprint in many cases (for example when it only needs to be known if the extent is increasing).

We conclude that the change identifiers are a useful and novel approach to the examination of dynamic dot patterns and that they have scope for use beyond that presented here. We make this statement while aware that there is certainly need for more testing over real world data and note with interest the forthcoming workshop to be chaired by Dr Antony Galton and Dr Zena Wood (Understanding and Modelling Collective Phenomena) which will hopefully provide more examples of such data.

10 Future Work

The examination presented in this thesis has focused primarily on the ability of change identifiers to reduce the number of required updates whilst tracking the footprint across a dynamic dot pattern. However it has also touched, albeit lightly, on other areas of research (mostly still within the field of spatio-temporal entities) in which they may provide benefit. This chapter is a look at these unexplored change identifier attributes with some preliminary thoughts and observations.

10.1 Dot Patterns

10.1.1 Dot Pattern Types

The descriptors presented in Chapter 3 provide a way of distinguishing different dot patterns, and it may be possible to use these differences to delineate between classes of dot patterns. Should there be intuitively distinct, and plausibly useful, classes that can be used to provide this sort of dot pattern taxonomy then the identifiers can be used to notify the user when a pattern switches from one type to another. The immediate difficulty faced by any taxonomy is in avoiding entirely arbitrary delineations between the pattern types, so to provide a defensible set of distinctions the values of the descriptors are conceptualised as vectors denoting a point in the 'descriptor space'. The Euclidean distance between these points provides a similarity measure and, with a sample set of randomised patterns, a clustering method (such as the agglomerative clustering method used as a descriptor) can be used to sort them into progressively larger clusters of similar patterns. By looking at the dendrogram[1] of this clustering a 'cut' can be made at the various levels allowing different clusterings of varying granularity to be selected.

Before any form of clustering can take place the descriptor values need to be normalised; without this step one descriptor can contribute more to the distance than another. A simple normalisation approach can be performed by taking the maximum and minimum values for a descriptor across the set of randomised patterns and scaling these to a range of -1 to 1. Descriptors that can result in values of infinity (gradient of principal component for example) are normalised across a a sigmoidal function curve as shown in Fig. 10.1.

Should the clustering analysis provide a set of clear descriptor divisions then the next task will be to decide how best to classify a new pattern without having to re-run the clustering process. A possible approach is to settle on some 'archetypal patterns' for each

[1] A graph with a tree-like structure showing the concatenation of clusters in the order they appear.

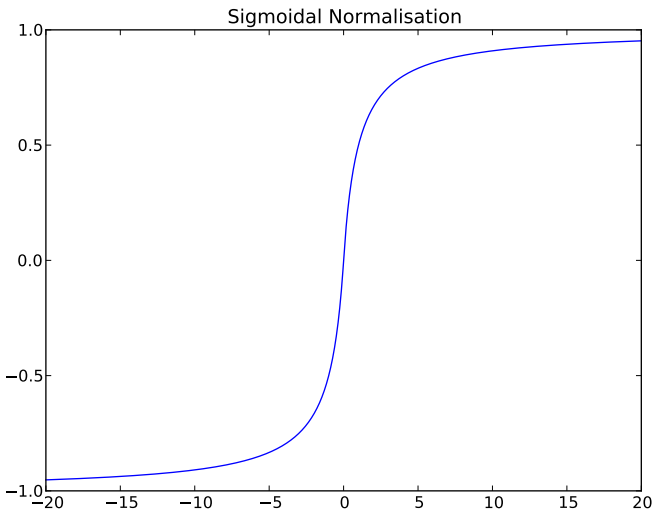


Figure 10.1 Graph showing Sigmoidal Normalisation Curve $\frac{x}{1+|x|}$.

of the delineations. The values of each archetype provide a central point in the descriptor space for their respective types. These points can be used to form a Voronoi division of the space and a pattern is therefore of the type whose archetype it is closest to. Such an approach allows a pattern from outside the test set of patterns used for clustering to be classified (the *training set*), but it requires that the descriptor values are normalised. The normalisation currently proposed relies on some maximum and minimum values for the descriptors (except those normalised on the sigmoidal curve), if the pattern to be classified (the *candidate* pattern) has values beyond the range of the maxima and minima of the training set then the clustering analysis will need to be re-run. To provide a classification that never requires the analysis to be re-run requires a training set that contains all possible extremal values for a descriptor; in effect every value would need to be normalised on a asymptotic curve like the sigmoidal approach discussed above. Even if the candidate's values are within the range of the training set, the classification of the candidate pattern is only relative to the training set. A classification that can delineate between the different types of dot pattern presented when tracking a herd's movement may be unsuitable for the set of dot patterns presented by the buildings within cities. A better approach would have sample patterns from the field in which we wish to classify the dot patterns and find the archetypes specific to that field.

Using an agglomerative clustering method we have clustered a randomised dynamic dot pattern of length 15 by the sigmoid normalised values of the fastest non-correlated descriptor set (given in Chapter 3) to give some preliminary indications as to whether or not such 'archetypal patterns' could be found. Only 15 patterns have been used as the graphs and associated image displays of larger sets are not particularly intelligible in a printed document.

The dendrogram in Fig. 10.2 shows a few large 'jumps' towards the end as the distance between the pattern clusters increases. At the cut off point indicated on the figure there are 3 clusters $\langle 5, 6\rangle$, $\langle 14, 1, 7\rangle$ and $\langle 0, 2 \to 4, 8 \to 13\rangle$, this cut-off was chosen as both 'legs'

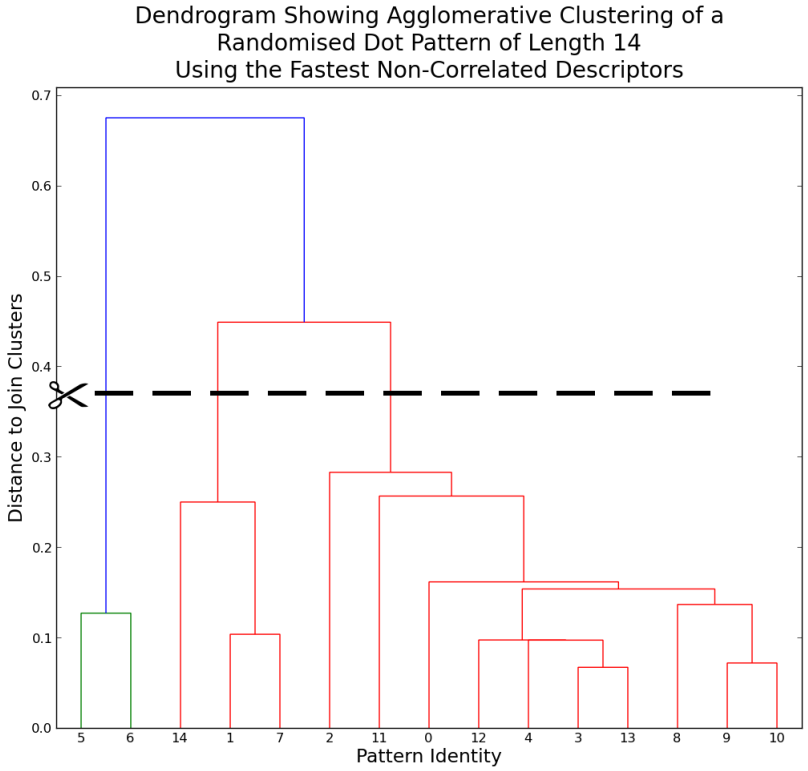


Figure 10.2 Dendrogram for the clustering of dot patterns by the fastest non-correlated descriptor set

of the branch joining $\langle 14, 1, 7\rangle$ to $\langle 0, 2 \rightarrow 4, 8 \rightarrow 13\rangle$ are of a significant length compared to the average 'jump'. The patterns defined by this division of the clustering can be seen in Fig. 10.3.

The class divisions at this clustering level are not necessarily those that a human may choose, but some of the reasons why the delineations have been made can be rationalised. For example, in the first cluster [5] and [6] are both mostly collinear patterns with similar orientation, while in the second cluster [7] and [1] also have similar orientation but in the opposite direction to cluster one. [14] does not have a particularly obvious orientation but it is, presumably, in the second cluster as it has similar connectedness and nearest neighbour variance to [1]. However not all the clusterings are as easily understood, for example why are [8] and [2] not connected until the join just before the cut-off shown in Fig. 10.2 despite both being very dense and comparatively small patterns? There are several situations that these non-intuitive clusterings may arise from:

- While the descriptors do not directly correlate, there may be tripartite correlation (as described in Chapter 3 and by Andrienko and Andrienko [4]) which unfairly weights some aspects of the clustering.

- The range of dot patterns used in the clustering is not wide enough and some descriptors are represented in a 'stronger' form than others. If the differences in extent are small compared to the differences in orientation then the orientation will have a larger weighting in the classification.

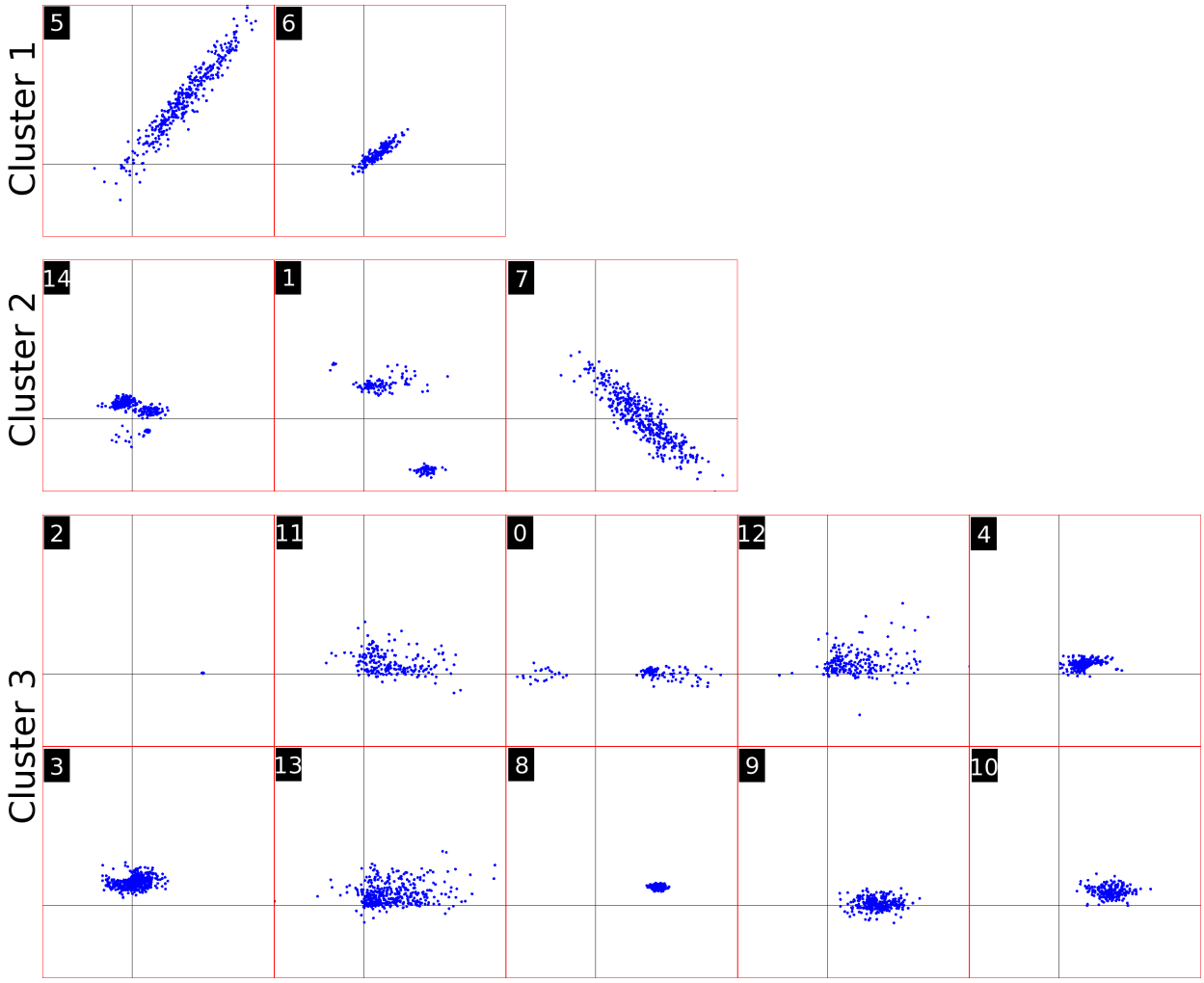


Figure 10.3 Snapshot of the clustering process of dot patterns by the fastest non-correlated descriptor set.

- The differences in the parameters that are not visually obvious outweigh the others.

Much more research needs to be performed before such a classification can be used, in particular the question of how the classification is assessed would need to be answered. For example, would the classification be better if the 'archetypal' dot pattern types/classes are those that are intuitive to a human? Answering such questions would require an in-depth analysis of the types of information required in applications using dot patterns and, given the range of fields in which such patterns appear, will likely have context specific answers.

Should definitive classes arise from the agglomerative clustering process it is will then be possible to find a best fit class for any given dot pattern. The dynamic dot pattern can be seen as a traversal across the 'class' space of dot pattern types. While we may expect a specific dynamic dot pattern to not move too far from its initial class it is certainly possible that it will migrate gradually towards other classes. For example a herd of wildebai[2] being chased by a predator may go from a single spread out grouping to several dense fleeing packs.

[2]One of three possible plurals for wildebeest: wildebeest, wildebeests and wildebai

10.1.2 Plotting a Dynamic Dot Pattern

A further abstraction on the descriptor concept is to imagine the dot patterns as existing in a multi-dimensional 'descriptor space', with each descriptor providing a coordinate axis. When mapped onto this space each dot pattern forms a single dot, with each of its descriptor values specifying a position on the appropriate axis. Plotting all of the phases of a single dynamic dot pattern produces a new dot pattern, which represents the dynamic dot pattern by the range of descriptor values it has undergone. The footprint of this resultant dot pattern describes the bounds of the dynamic dot pattern with respect to the descriptors and is, therefore, representative of the limits of the change of the underlying collective[3]. If time is added to the axes of the descriptor space then, instead of producing a footprint, the dynamic dot pattern traces a path through its descriptor values. Using the descriptor space to produce a footprint or a path for a dynamic dot pattern may provide a new and interesting approach to classifying the collectives by their behaviour.

10.2 Footprints

10.2.1 Footprint Types

We can examine the footprint at each time step at which it is updated to ascertain its intrinsic type within the classification presented in Chapter 4. However, unlike the dot pattern types suggested above, the calculations are outside the scope of the change identifiers – which only examine the dot patterns. Instead the calculations must be added as a new module to the framework; increasing the time taken for computation at each time step that the footprint is updated. Preferably this increase in time will not void the gain made in using change identifiers. The state of all of the intrinsic footprint criteria can be found by iterating over the edges of the footprint, and are therefore of at least time complexity $O(v)$ in which v is the number of vertices of the footprint. This increase in calculation cost may be allowable by imagining that the time saved by the change identifiers can be 'spent' on other tasks, as long as the system as a whole can still provide a more up-to-date footprint than attempting to update at each time step. An alternative way in which to consider the cost-benefit trade-off is that the time lost while performing the footprint classification can be allowed if the average calculation time per timestep when using identifiers and footprint classification is less than the average calculation time per time step when updating at each timestep.

Being able to indicate to a user when the footprint type changes is of benefit for the same reason that the dot pattern type change notification would be; it provides further information about the nature of the change that the collective is undergoing. In addition to the footprint state, we may also be able to use this information to indicate when the footprint algorithms parameter is no longer suitable.

[3] Assuming we have descriptors that accurately capture the state of the collective.

10.2.2 Footprint Algorithm Parameters

In the background chapter, it was noted that most footprint algorithms need a user defined parameter. The parameter requirement led to the discussion within the results chapter of the reasoning behind the algorithm parameter selection used in our experimentation, which aimed primarily to provide fair testing conditions. There is still much work to be done on the informed selection of the footprint algorithm parameters; the work presented in this thesis on this area being very much inchoate. The use of the footprint classification may well help with identifying when the parameter need change but does not provide a starting value. Using the footprint classification to select parameters is further hamstrung by the, possibly erroneous, assumptions that the parameter needs to be changed when the footprint type changes and that this is the only point at which it need change.

To attain a better understanding of the parameter and footprint relation will likely require an in-depth examination of the interplay between the dot pattern types and the footprint algorithms. The footprint algorithm classification (Chapter 4) will need to be extended to include a description of the nature of the parameter. For example if the algorithm uses the parameter as a threshold on the edge length (Swinging Arm [28], χ-hull [20]) it will require a different starting parameter than that of an algorithm which requires the number of neighbours to compare at each iteration (K-Nearest Neighbours [50]).

10.3 Change Identifiers

There is further research that can be performed on the information provided by identifiers beyond using them to update footprints. The identifiers provide immediate information about the fashion in which the dynamic dot pattern is changing, and this information leads to a description of the complex behaviours the underlying collective phenomenon of the pattern is undergoing. The changes themselves tend to be small and are increments or decrements in quantitative values but this thesis has previously suggested that it may be possible to provide qualitative information directly rather than having a human observer interpret the results. It may be that, for some applications, providing information about the expansion of a dynamic dot pattern via a status update (e.g. “The phenomenon is expanding by 10% every 20 seconds”) is more useful than showing the expansion of a footprint. Change identifiers can be used to bypass the footprint entirely by relaying only the useful change information; reducing the amount of assessment needed to be performed by the user.

It should also be noted that the change identifiers presented in this thesis do not exhaust the range of possible identifiers, and there are almost certainly other useful measurements to be added to the set we provide. They could also be further examined by looking for three-way correlation of they type suggested by Andrienko and Andrienko structure consideration [4] and to see if the difference between the first and second order effects proposed by Sullivan and Unwin can be identified [51].

10.4 Other Fields

The work in this thesis has primarily focused on examples that are spatio-temporal in nature; situating them firmly in the area of GISc. However, as mentioned in the introduction, there are other fields which use data that can be visualised as dynamic dot patterns. For example the movement through the solution space of an optimisation problem or a set of of changing entities in some classification space. It would be interesting to see if the application of both footprints and change identifiers to these fields provides new insights into their behaviours.

Bibliography

[1] A.G.Hamilton. *Linear Algebra – An Introduction With Concurrent Examples*. Cambridge University Press, 1994.

[2] 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.

[3] Walid Ali and Bernard Moulin. 2d-3d multiagent geosimulation with knowledge-based agents of customers shopping behavior in a shopping mall. In AnthonyG. Cohn and DavidM. Mark, editors, *Spatial Information Theory*, volume 3693 of *Lecture Notes in Computer Science*, pages 445–458. Springer Berlin Heidelberg, 2005.

[4] Natalia Andrienko and Gennady Andrienko. Designing visual analytics methods for massive collections of movement data. *Cartographica*, 42(2):117–138, 2007.

[5] 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.

[6] Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, SODA '97, pages 747–756, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[7] R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1:173–189, 1972.

[8] Marc Benkert, Joachim Gudmundsson, Florian Hübner, and Thomas Wolle. Reporting flock patterns. *Computational Geometry - Theory and Applications*, 2007.

[9] Brandon Bennett, Derek R. Magee, Anthony G. Cohn, and David C. Hogg. Enhanced tracking and recognition of moving objects by reasoning about spatio-temporal continuity. *Image and Vision Computing*, 26(1):67–81, January 2008.

[10] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry, Algorithms and Applications*. Springer, third edition, 2008.

[11] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[12] D. Black. *Investigation of the possible increased incidence of cancer in West Cumbria: report of the Independent Advisory Group*. H.M.S.O., 1984.

[13] P. Bogaert, N. Van de Weghe, A.G. Cohn, F. Witlox, and P. De Maeyer. Reasoning about moving point objects on networks. In M Raubal, J H Miller, U A Frank, and F Goodchild, editors, *4th International Conference on Geographic Information Science (GIScience 2006)*, 2006.

[14] 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.

[15] Yi-Jen Chiang and Roberto Tamassia. Dynamic algorithms in computational geometry. In *Proceedings of the IEEE*, volume 80, pages 1412–1434, 1992.

[16] Kalyanmoy Deb. *Multi-Objective Optimization using evolutionary Algorithms*. Wiley, 2001.

[17] Géraldine Del Mondo, John G. Stell, Christophe Claramunt, and Rémy Thibaud. A graph model for spatio-temporal evolution. *Journal of Universal Computer Science*, 16(11):1452–1477, 2010.

[18] Matthias Delafontaine, Anthony G. Cohn, and Nico Van de Weghe. Implementing a qualitative calculus to analyse moving point objects. *Expert Systems with Applications*, 38(5):5187–5196, 2011.

[19] Somayeh Dodge, Robert Weibel, and Anna-Katharina Lautenschtz. Towards a taxonomy of movement patterns. *Information Visualization*, 7(3-4):240–252, 2008.

[20] 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.

[21] Max Dupenois and Antony Galton. Assigning footprints to dot sets: An analytical survey. In K. S. Hornsby, C. Claramunt, M. Denis, and G. Ligozat, editors, *Spatial Information Theory: Proceedings of the 9th International Conference COSIT 2009*, pages 227–244, Berlin, 2009. Springer.

[22] H. Edelsbrunner. Weighted alpha shapes. Technical Report UIUCDCS-R-92-1760, Department of Computer Science, University of Illinois, 1992.

[23] 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.

[24] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. In *Proceedings of the 1992 workshop on Volume visualization*, VVS '92, pages 75–82. ACM, 1992.

[25] Max Egenhofer and Robert Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2):161–174, 1991.

[26] Antony Galton. *Qualitative Spatial Change*. Oxford University Press, 2000.

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

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

[29] Yossi Gofman. Outline of a set of points. *Pattern Recognition Letters*, 14(1):31–38, 1993.

[30] Christopher M. Gold. Data structures for dynamic and multidimensional gis. In *4th ISPRS Workshop on Dynamic and Multi-dimensional GIS*, pages 36–41, Pontypridd, Wales, UK, 2005.

[31] D E Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[32] Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4):132–133, 1972.

[33] Leonidas Guibas. Kinetic data structures. In D. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*, pages 23–1–23–18. Chapman and Hall/CRC, 2004.

[34] Leonidas Guibas, Menelaos Karaveles, and Daniel Russel. A computational framework for handling motion. In *Proceedings of teh Sixth Workshop on Algorithm Engineering and Experiments*, pages 129–141, 2004.

[35] Leonidas J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. *Foundations of Computer Science, IEEE Annual Symposium on*, 0:8–21, 1978.

[36] John Hershberger and Subhash Suri. Convex hulls and related problems in data streams. In *Proceedings of ACM/DIMACS Workshop on Management and Processing of Data Streams*, pages 148–168, 2003.

[37] Kathleen Hornsby and Max J. Egenhofer. Qualitative representation of change. In S. Hirtle and A. Frank, editors, *Spatial Information Theory: A Theoretical Basis for GIS, Proceedings of the International Conference COSIT'97*, pages 15–33. Springer-Verlag, 1997.

[38] Yan Huang, Cai Chen, and Pinliang Dong. Modeling herds and their evolvements from trajectory data. In *GIScience '08: Proceedings of the 5th international conference on Geographic Information Science*, pages 90–105, Berlin, Heidelberg, 2008. Springer-Verlag.

[39] 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.

[40] Jixiang Jiang and Michael Worboys. Detecting basic topological changes in sensor networks by local aggregation. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, GIS '08, pages 4:1–4:10, New York, NY, USA, 2008. ACM.

[41] Jixiang Jiang, Michael Worboys, and Silvia Nittel. Qualitative change detection using sensor networks based on connectivity information. *GeoInformatica*, 15:305–328, 2011.

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

[43] Donald Knuth. *The Art Of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, 2nd edition, 2007.

[44] P. Laube, M. Van Kreveld, and S. Imfeld. Finding remo - detecting relative motion patterns in geospatial lifelines. In P. F. Fisher, editor, *Developments in Spatial Data Handling: Proceedings of the 11th International Symposium on Spatial ata Handling*, pages 201–214. Springer, 2004.

[45] Patrick Laube, Matt Duckham, and Marimuthu Palaniswami. Deferred decentralized movement pattern mining for geosensor networks. *International Journal of Geographical Information Science*, 25(2):273–292, 2011.

[46] Patrick Laube and Ross S. Purves. How fast is a cow? cross-scale analysis of movement data. *Transactions in GIS*, 15(3):401–418, 2011.

[47] Mahmoud Melkemi. $\mathcal{A}$-shapes of a finite point set. In *Proceedings of the thirteenth annual symposium on Computational geometry*, SCG '97, pages 367–369. ACM, 1997.

[48] Mahmoud Melkemi and Mourad Djebali. Computing the shape of a planar points set. *Pattern Recognition*, 33(9):1423 – 1436, 2000.

[49] Mahmoud Melkemi and Mourad Djebali. Weighted $\mathscr{A}$-shape: a descriptor of the shape of a point set. *Pattern Recognition*, 34(6):1159 – 1170, 2001.

[50] 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.

[51] David O'Sullivan and David J. Unwin. *Geographic Information Analysis.* Wiley, November 2002.

[52] Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Science*, 23(2):166–204, 1981.

[53] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 25–34, New York, NY, USA, 1987. ACM.

[54] P.L. Rosin. Measuring shape: ellipticity, rectangularity, and triangularity. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 1, pages 952 –955 vol.1, 2000.

[55] P. H. Sneath. The application of computers to taxonomy. *J. Gen. Microbiol.*, 17:201–226, Aug 1957.

[56] N. Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[57] John G. Stell. Granularity in change over time. In M. Duckham, M. Goodchild, and M Worboys, editors, *Foundations of Geographic Information Science*, chapter 6, pages 95 – 115. Taylor and Francis, 2003.

[58] Marius Thériault, Christophe Claramunt, and Paul Villeneuve. A spatio-temporal taxonomy for the representation of spatial set behaviours. In Michael Bhlen, Christian Jensen, and Michel Scholl, editors, *Spatio-Temporal Database Management*, volume 1678 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin / Heidelberg, 1999.

[59] J. Žunić and P.L. Rosin. A convexity measurement for polygons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:173–182, 2002.

[60] J. Žunić and P.L. Rosin. Rectilinearity measurements for polygons. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(9):1193 – 1200, September 2003.

[61] Zena Wood. *Detecting and Identifying Collective Phenomena within Movement Data*. PhD thesis, University of Exeter, 2011.

[62] Zena Wood and Antony Galton. A taxonomy of collective phenomena. *Applied Ontology*, 4:267–292, August 2009.

[63] Michael Worboys. Event-oriented approaches to geographic phenomena. *International Journal of Geographical Information Science*, 19:1–28, 2005.

[64] Michael Worboys and Matt Duckham. *GIS: A Computing Perspective*, chapter 6.4 Point Object Structures, pages 240 – 248. CRC Press, 2nd edition, 2004.