A New Method for Semi-Supervised Density-Based Projected Clustering

by

Zachary Jullion

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Density-based clustering methods extract high density clusters which are separated by regions of lower density. HDBSCAN* is an existing algorithm for producing a density-based cluster hierarchy. To obtain clusters from this hierarchy it includes an instance of FOSC (Framework for Optimal Selection of Clusters) to extract significant clusters, based on a measure known as cluster stability. We introduce CASAR (Compact And Separation Adjusted Ratio), a new algorithm for extracting significant clusters from an HDBSCAN* hierarchy. CASAR is similar to FOSC, but defines local cluster quality differently and also uses a different aggregation method for comparing the quality of descendant clusters to ancestors in the hierarchy. The local cluster quality that CASAR uses is based on the validation index DBCV (Density-Based Cluster Validation). CASAR is designed to extract individual density-based clusters from subspaces, and is not meant to be a general purpose replacement for cluster stability.

We also introduce a new semi-supervised density-based method for finding relevant subspaces. Given a set of should-link objects that belong to an undiscovered cluster, our method finds an appropriate set of attributes for extracting the cluster. Our method makes use of well-established qualities of density-based clusters, and as such, it can be used as a pre-processing step for a wide variety of different density-based clustering algorithms. We combine this method with HDBSCAN* and CASAR to produce a semi-supervised density-based projected clustering algorithm.

In a series of experiments, we compare CASAR and cluster stability on both synthetic data and on real data sets. We also compare our semi-supervised density-based projected clustering algorithm to an existing semi-supervised projected clustering algorithm and to a well-

known unsupervised projected clustering algorithm. We conclude this thesis with a summary of the strengths and weaknesses of our method, a summary of experimental findings, and a discussion about possible directions for future work.

Acknowledgements

I would like to express immense gratitude to my supervisor, Joerg Sander, for his guidance, expertise, and patience. This thesis would certainly not have been completed without him. His considerable experience in the field of KDD was invaluable, as was his understanding of the complexities of academic research.

Thanks to the other two members of my supervisory committee, Ricardo Campello and Mario Nascimento. I am thankful to Ricardo for his continued interest in my research, advice, and willingness to be on this committee. Mario supervised me through two summer NSERC research scholarships when I was an undergraduate student, and I am very grateful for his presence on this committee as well.

Thanks to Davoud Moulavi for his advice and help in researching various clustering methods.

Thanks to Arthur Zimek for his interest in my research and for his advice.

Table of Contents

List of Figures

Chapter 3:

Chapter 4:

Chapter 5:

Chapter 6:

List of Tables

Chapter 4:

Chapter 6:

Chapter 1: Introduction

As computers become increasingly more powerful and less expensive, digital information is becoming both easier to obtain and less difficult to store. Data sets that at one time would have been considered nearly impossible to collect and digitize are now commonplace. Consider an everyday example of such data collection and storage: social media. Large social media platforms can easily store hundreds of pieces of information on millions of people. In addition to social media platforms, massive data sets can be collected through wireless sensor networks, image repositories, microarrays, traffic sensors, point-of-sale software, etc. It is apparent that we live in a world of 'big data': one in which an incomprehensible amount of digital information is being captured and stored at any given moment.

In many fields of research, the challenge of 'big data' is finding useful information (that is, useful patterns or new forms of knowledge) in massive data sets. Generally speaking, the process of analyzing a collection of data in order to obtain useful information from it is called knowledge discovery in databases (KDD) [Fayyad et al, 1996]. Fayyad, Piatetsky-Shapiro, and Smyth view KDD as the selection, pre-processing, transformation, mining, and interpretation/evaluation of data [Fayyad et al, 1996]. Note that data acquisition is not a part of the KDD process. Similarly, the application of knowledge is also not a part of the KDD process. KDD is strictly concerned with finding useful information in an existing data set, without specifying how data should be obtained, or how the acquired knowledge should be used.

Obviously, manually analyzing a large data set (even such as one with only thousands of objects) is practically infeasible. In order to facilitate KDD on such data sets, a variety of algorithms have been developed. One class of such algorithms are clustering algorithms. Clustering is a part of the data mining step in KDD [Fayyad et al, 1996]. Jain, Murty, and Flynn define clustering as the “unsupervised classification of patterns (observations, data items, or feature vectors) into groups (clusters)” [Jain et al., 1999]. Typically, clustering algorithms partition the objects in a data set into several clusters based on some measure of similarity. Some objects may also be assigned to the 'noise' group, indicating that they do not belong to any cluster.

There are a wide variety of different types of clustering algorithms. A large part of the reason why many different clustering algorithms exist is that there are many different types of

data sets. Some data sets are spatial in nature, while others are represented as graphs, while still others track data changing over time. Some data sets are challenging to cluster because they contain so many entries, while for others the challenge may be the large number of attributes per entry. Unfortunately, there is no general clustering algorithm that can correctly partition all data sets, and as such, new clustering algorithms are continually needed.

For several years, the 'curse of dimensionality' has been a prominent problem being addressed by the clustering community. [Aggarwal et al., 2001, Beyer et al., 1999, Hinneburg et al., 2000]. Moise et al. state that "increasing data dimensionality results in the loss of contrast in distances between data points. Thus, clustering techniques that measure the similarity between data points by considering all attributes of a data set tend to break down in high-dimensional spaces." [Moise et al., 2009] Essentially, as more and more attributes are added to a data set, the distances between objects become increasingly similar, and as such, objects can no longer be partitioned into separate clusters. For most algorithms that consider the full set of attributes when clustering, this means that, at some point, adding information in the form of additional attributes can lead to a worse (less correct) clustering result, even though this may seem counter-intuitive.

Subspace and projected clustering algorithms exist to overcome the problems presented by the curse of dimensionality. Although these two types of approaches have some key differences, they both find clusters in subsets of the full set of attributes. Subspace and projected clustering algorithms do not just partition the set of objects in the data set into a set of clusters: they also assign sets of attributes to each cluster. These types of algorithms assume that each cluster is only valid in a given set of attributes, and that all attributes that are not valid for a given cluster are noise, and should be ignored for that cluster. In many cases, these assumptions are valid. Certain attributes in a data set may not help differentiate any of the clusters. This can occur because the process of data collection was flawed, or simply because the attribute that was collected is not meaningful in any significant way. Furthermore, some attributes may help differentiate certain clusters, but not contribute to other clusters.

Semi-supervised clustering algorithms are another relatively recent development in clustering. These types of algorithms make use of user provided constraints in order to improve the quality of the partition. There are many different types of constraints, each of which provides a different type of information. Two of the most common constraints are should-link constraints and should-not-link constraints. Should-link constraints indicate that two (or sometimes more)

objects in the data set belong to the same cluster, while should-not-link constraints indicate that two (or sometimes more) objects in the data set do not belong to the same cluster. Constraints are leveraged in many different ways across different algorithms, but in almost all cases, having even a small number of user provided constraints significantly improves the clustering result.

In addition to classifying clustering algorithms as subspace, projected, or traditional (operating in the full set of attributes), and semi-supervised or unsupervised (not making use of constraints), algorithms can also be classified as partitional or hierarchical. Specific details on the differences between partitional and hierarchical methods will be described later in the thesis. For now, it is important to note that this thesis focuses on a type of partitional clustering called 'density-based clustering', and that the objective of density-based algorithms is to find clusters of high density separated from each other by regions of low density. Density-based clustering algorithms are able to find clusters of arbitrary shape in a data set.

This thesis presents semi-supervised density-based methods for dealing with the 'curse of dimensionality'. Specifically, we present a method for finding a relevant set of attributes (known as a ‘subspace’) to perform clustering in, and we adapt an existing methodology for density-based clustering to find single clusters in relevant subspaces. Given a set of should-link objects (that is, a set of objects that belong to the same cluster), we can find the corresponding density-based cluster embedded in a relevant subspace. To the best of our knowledge, at the time of release, this is the only semi-supervised density-based projected clustering algorithm that exists. It should be noted that there are many other algorithms which combine two of these three paradigms (semi-supervised, density-based, and projected). Additionally, SC-MINER [Fromont et al., 2009] is highly related, as it is a semi-supervised density-based subspace clustering algorithm.

The remainder of this thesis is laid out as follows:

In Chapter 2, related works are discussed. A variety of density-based, subspace, and semi-supervised clustering algorithms are examined. Some of these algorithms are over a decade old and help to form the basis for our new methods, while others are more recent and can better be seen as competitive alternatives to our methods. Additionally, the terms 'density-based', 'projected', and 'semi-supervised' are briefly explained in Chapter 2. In particular, the differences between partitional and hierarchical algorithms, subspace and projected algorithms, and semi-supervised and unsupervised algorithms are explored. These differences are highlighted by the

variety of related works that are examined.

Chapter 3 introduces the basic concepts of density-based clustering. A variety of definitions and ideas are explained in detail. We then explain how HDBSCAN* produces a density hierarchy, and how FOSC uses the cluster stability measure to extract a flat partition from this hierarchy [Campello et al., 2013a, Campello et al., 2013b, and Campello et al., 2015]. Additionally, we briefly explain how FOSC can leverage user provided constraints.

In Chapter 4, we introduce CASAR. CASAR can extract a flat partition from an HDBSCAN* hierarchy in a manner similar to FOSC. Unlike FOSC, CASAR is specifically intended for extracting a single density-based cluster from a relevant subspace. We explain the theory behind CASAR and then compare CASAR both conceptually and experimentally to the cluster stability measure that FOSC uses. A wide variety of experiments are run on synthetic (artificially generated) and real data sets.

In Chapter 5, we present a method for semi-supervised attribute selection – in other words, a method for finding a relevant subspace for a density-based cluster. Note that this density-based cluster has not yet been found – it will be extracted after the relevant subspace is constructed. We also explain how our method can be combined with HDBSCAN* and CASAR to produce a semi-supervised density-based projected clustering algorithm. This algorithm takes as input sets of should-link objects and finds a relevant subspace and a density-based cluster for each such set.

We compare the performance of our semi-supervised density-based projected clustering algorithm to several existing algorithms in Chapter 6. Some experiments are run on synthetic data sets, while other experiments are run on modified versions of real data sets.

In Chapter 7, we conclude the thesis. We summarize the strengths and weaknesses of our method, summarize our experimental findings, and provide possible directions for future work.

Chapter 2: Related Work

In this Chapter, the basis for density-based clustering, semi-supervised clustering, and projected clustering will be examined. First, the differences between two major clustering paradigms – partitional and hierarchical – will be explained. Next, density-based clustering will be discussed. Some of the strengths of density-based clustering will be presented, and key density-based algorithms will be summarized. Next, semi-supervised clustering will be examined. Several different types of constraints and semi-supervised clustering algorithms will be discussed. Then, subspace and projected clustering will be examined. The differences between these two paradigms will be explained, and important subspace and projected algorithms will be summarized. Finally, semi-supervised projected clustering methods and related methods (such as semi-supervised subspace clustering methods and semi-supervised dimensionality reduction techniques) will be examined.

2.1 Clustering Paradigms

The two most commonly recognized clustering paradigms are partitional and hierarchical. Jain et al. succinctly summarize the difference between these two paradigms when they state that "hierarchical methods produce a nested series of partitions, while partitional methods produce only one" [Jain et al., 1999]. As the name suggests, partitional clustering algorithms partition the data set – that is, they divide the data set into a number of clusters. In most partitional methods, each object in the data set is assigned to a single cluster. Hierarchical clustering algorithms, on the other hand, produce a cluster tree known as a *dendrogram*. A dendrogram consists of a single root cluster that contains all points, which is gradually split into smaller and smaller clusters in a hierarchical manner.

A wide variety of partitional clustering algorithms exist. For instance, *K-means* and *k-medoid* algorithms operate in an iterative manner. Both require the number of clusters as a user-defined parameter. Typically, these types of algorithm begins with a set of cluster 'seeds' (objects that represent the centroid or median of each cluster). Different algorithms may have different processes for selecting or generating this set of seeds. An iterative process of assigning objects to clusters and re-assigning seeds based on the new clusters is then repeated, until a stopping

criterion is reached [Jain et al., 1999, Omran et al., 2007]. The *EM (Expectation-Maximization)* clustering algorithm "partitions the data set into clusters by determining a mixture of Gaussians fitting the data set. Each Gaussian has a mean and covariance matrix" [Omran et al., 2007]. A variety of density-based partitional methods also exist, such as DBSCAN [Ester et al., 1996] and DENCLUE [Hinneburg & Keim, 1998].

Hierarchical clustering algorithms "generate a cluster tree (or dendrogram) by using heuristic splitting or merging techniques" [Omran et al., 2007]. These types of algorithms are either divisive (starting with a single cluster consisting of all objects, and then splitting this cluster into smaller clusters) or agglomerative (starting with many small clusters – potentially even having each object as its own cluster – and then merging these smaller clusters until all objects are part of one cluster). Rather than returning a single set of clusters, hierarchical clustering algorithms return a cluster tree which contains many different partitions at different 'levels'. A variety of different types of hierarchical clustering algorithms exist, including density-based ones such as HDBSCAN [Campello et al., 2013, Campello et al., 2015]. It is worth noting that hierarchical algorithms are often slower than partitional algorithms, because producing a dendrogram that consists of many partitions at many levels is often more time-consuming than producing a single, 'flat' partition.

It is important to note that there are other ways of classifying clustering algorithms, beyond labelling them as 'partitional' or 'hierarchical'. Jain et al. provide several examples of other dichotomies that exist when classifying clustering methods [Jain et al., 1999]. For example, some methods produce *hard* partitions, while others are *fuzzy*. Hard clustering algorithms only allow each object in the data set membership in a single cluster, while fuzzy algorithms may allow each object partial membership in many different clusters. *Deterministic* methods will always produce the same results, given the same input, while *stochastic* methods introduce an element of randomness, and hence the same inputs can produce slightly (or, in rare cases, extremely) different results across different runs. A wide variety of other classification schemes for clustering algorithms also exist, but will not be further explored in this thesis.

2.2 Density-Based Clustering

In his 1975 book, *Clustering Algorithms*, Hartigan provides a theoretical basis for density-based clustering when he explains density-contour trees [Hartigan, 1975]. Hartigan begins this

explanation by noting that "clusters may be thought of as regions of high density separated from other such regions by regions of low density" [Hartigan, 1975]. He goes on to say that "It is easiest to first formalize this definition for a distribution of points (consisting of an infinite number of points), characterized by a density $f(\boldsymbol{x})$ at each point $\boldsymbol{x}$. The number $f(\boldsymbol{x})$ is proportional to the number of objects per unit volume at the point $\boldsymbol{x}$. A *density-contour* cluster at level f_0 is a subset C of the N-dimensional space, such that C is maximal among connected sets satisfying $f(x) \geq f_0$ for $x \in C$" [Hartigan, 1975]. Essentially, a density-contour cluster is a connected region of high density in the data set (where 'high' here means that at any point in the cluster, the density is greater than some level f_0). Hartigan notes that by reducing the level f, clusters will gradually grow and eventually join with other clusters (conversely, increasing the level f will cause clusters to shrink and eventually split into smaller clusters or disappear entirely). Although the definitions just given are for a data set with an infinite number of objects, Hartigan explains that it is trivial to expand these definitions to data sets with a finite number of objects.

When explaining their definition of density-based clusters, Ester et al. state that "within each cluster we have a typical density of points which is considerably higher than outside of the cluster. Furthermore, the density within the area of noise is lower than the density in any of the clusters" [Ester et al., 1996]. Although different density-based clustering algorithms may have different ways of computing density, different criteria for separating clusters, and/or different intended applications, the notion that clusters are regions of high density separated by regions of low density is one that all density-based clustering algorithms share.

One of the first (and most influential) density-based clustering algorithms is DBSCAN (density-based spatial clustering of applications with noise) [Ester et al., 1996]. In their paper, Ester et al. present several concepts that are key to density based clustering. First of all, they introduce the notion of *core points*. A core point is an object in a data set which has at least *MinPts* objects within a distance of *Eps* (the *Eps-neighborhood*) of itself. As well, note that both *MinPts* and *Eps* are user-defined parameters, and that any distance function can be used. Core points are the 'dense' objects in a data set, and will therefore be members of clusters. *Border points* will also be clustered. These objects are not dense like core points (they have less than *MinPts* objects in their Eps-neighborhood), but they are within a core point's Eps-neighborhood.

The second important definition that Ester et al. provide is that of *density-connectivity.*

Ester et al. begin by explaining that an object *A* is *directly density-reachable* from an object *B* if *B* is a core point and *A* is within *B*'s Eps-neighborhood. Note that *B* is only directly-density-reachable from *A* if *A* is also a core point. Next, they define two objects as being *density-reachable* if a chain of objects, each directly density-reachable from the previous object in the chain, exists between two objects. Again note that, like the definition of directly density-reachable, density-reachable is not not symmetric when the last object in the chain is not a core point. Finally, Ester et al. define two objects *A* and *B* as being *density-connected* if there is a third object, *C*, from which both *A* and *B* are density-reachable. The density-connected relationship is always symmetric.

Ester et al. provide their third important concept when they define a *density-based cluster*. A density-based cluster is a set of objects which are all density-connected (given a certain *Eps* and *MinPts*). As well, any object that is density-reachable from any object in the cluster must also be a member of the cluster (that is, the cluster is maximal). Essentially, each cluster consists of one or more density-connected core points, as well as any border points within the *Eps-neighborhood(s)* of these core point(s). Note that each cluster will contain at least *MinPts* objects, as each cluster must minimally have a core point and its *Eps-neighborhood*. All objects in a data set which are not part of a cluster are classified as noise.

DBSCAN finds density-based clusters in a data set through an iterative process. For each unclassified object in the data set, DBSCAN determines how many points are in the Eps-neighborhood of the object. If there are less than *MinPts* objects in the Eps-neighborhood, then the object currently being examined is classified a noise object, and the process continues onto the next unclassified object. Otherwise, if there are *MinPts* or more objects in the Eps-neighborhood, the object being examined is a core point, and a new cluster has been found. All objects within the Eps-neighborhood of this core point will be added to the cluster, and will then be iteratively examined, to determine if any of them are also core points. Any core points found in this manner will have the same iterative process repeated on them, until all of the cluster's core points have been found. A final cluster (consisting of core points and border points) is then saved, and DBSCAN continues on to the next unclassified object in the data set.

OPTICS is a density-based cluster analysis tool that borrows many concepts from DBSCAN. This algorithm “does not produce a clustering of a data set explicitly; but instead creates an augmented ordering of the database representing its density-based clustering structure.

This cluster-ordering contains information which is equivalent to the density-based clusterings corresponding to a broad range of parameter settings" [Ankerst et al., 1999]. Essentially, for some setting of *MinPts* and *Eps*, OPTICS traverses the data set object by object, searching for directly-density-reachable 'chains' of core objects. OPTICS always travels from the current object to the next unprocessed object which is both dense and close to the current object (that is, the most strongly density-reachable object). In this manner, OPTICS is able to construct a *reachability plot* which visualizes the reachability distances between objects in the data set. Clusters appear as continuous 'low' regions in this visualization (indicating that there are a number of dense objects which are all density-connected), while noise objects appear as high peaks (for a small number of noise objects) or high regions (for a large number of noise objects that were processed sequentially).

DENCLUE is a density-based clustering algorithm that is "based on the idea that the influence of each data point can be modeled formally using a mathematical function" [Hinneburg & Keim, 1998]. This function is called an *influence function*. A variety of different influence functions (such as square wave and Gaussian) can be used. This algorithm finds arbitrarily-shaped dense clusters in a data set by searching for local maxima in the influence function using a hill-climbing procedure. These maxima represent the most dense points in the data such, and as such, are translated into clusters (along with relevant nearby points). DENCLUE has two input parameters: σ and ξ. σ affects how much influence a point exerts, while ξ is a threshold value that indicates whether a point is a local maximum or not. It is important to note that, as a preclustering step to improve efficiency, DENCLUE constructs a number of hypercubes of the data, and then only processes hypercubes which contain more than a certain threshold number of objects. This preclustering step may sometimes cause relevant cluster to be filtered out.

A more recent example of a density-based clustering algorithm is HDBSCAN* (Hierarchical DBSCAN*) [Campello et al., 2013, Campello et al., 2015]. As the name suggests, HDBSCAN* finds density-based clusters in a hierarchy. One of the reasons that HDBSCAN* is of interest is that it redefines what a density-based cluster is. Specifically, HDBSCAN* introduces the DBSCAN* algorithm. While DBSCAN returns clusters that consist of core points and corresponding border points, clusters found by DBSCAN* consist of core points only. Campello et al. note that this change is made because "border objects do not technically belong to the level set (their estimated density is below the threshold)" [Campello et al., 2015].

HDBSCAN* also introduces an additional change: the introduction of another user-defined parameter, *minClSize*. The *minClSize* parameter specifies the minimum number of density-connected core points that are required to form a cluster. With the introduction of *minClSize*, some core points may no longer be part of a cluster.

HDBSCAN* produces a *dendrogram* such that the root of the cluster tree is a single low-density cluster consisting of all objects in the data set, while the leaves of the cluster tree are the smallest possible high-density clusters (see section 3.2 for a description of this process). The minimum density required to form a cluster increases as the dendrogram is traversed from the root to the leaves, and as such, clusters become increasingly smaller, sometimes splitting into several clusters or disappearing altogether. Obviously, the opposite effects occur as the dendrogram is traversed from the leaves to the root. This dendrogram provides each unique partition of the data set that would be possible when running DBSCAN as the *Eps* parameter (essentially, the minimum density required to form a cluster) is adjusted. It should be noted that HDBSCAN* is capable of extracting a single 'flat' partition from this dendrogram. HDBSCAN* is able to make use of constraints when extracting such a flat partition, and as such, the algorithm is semi-supervised.

All of the density-based algorithms previously mentioned suffer from several drawbacks. First of all, most have several input parameters that are difficult to set. DBSCAN has the input parameters *Eps* and *MinPts*. The relationship between these parameters is not immediately obvious, and, even for domain experts, setting these parameters correctly for a given data set may prove to be extremely difficult. Similarly problems arise for the input parameters σ and ξ used by DENCLUE. Secondly, both DBSCAN and DENCLUE are only able to find clusters at a single density level: that is, they use a global density threshold, and as such, several dense clusters which conceptually should be separated may often by combined into a single cluster (for a given density threshold). Increasing the density threshold (thereby separating these dense clusters) may cause many less dense clusters to disappear entirely, as they become noise. Thirdly, none of these algorithms (DBSCAN, OPTICS, DENCLUE, HDBSCAN*) are able to handle the curse of dimensionality. In data sets with a large number of attributes, each of these algorithms will have trouble differentiating distances between objects, and they will be unable to find clusters that exist in subsets of the full set of attributes. Finally, none of these algorithms (except HDBSCAN*) are able to take advantage of constraints that domain experts may be able to

provide. For this reason, semi-supervised clustering algorithms exist.

2.3 Semi-Supervised Clustering

Supervised learning, semi-supervised learning, and unsupervised learning are all different types of machine learning. Generally speaking, machine learning can be seen as the task of automatically (algorithmically) obtaining useful information (which was defined in Chapter 1 as useful patterns or new forms of knowledge) from data. As such, many tasks in the process of KDD can be seen as machine learning tasks (including clustering). Roughly speaking, these three different types of learning (supervised, semi-supervised, and unsupervised) differ in the amount of labelled input data they require/use, and the way(s) in which they use this labelled input data.

Supervised methods develop a model of the data by examining labelled *training data*. This model can then be used to interpret unlabelled objects. As an example of supervised learning, consider a classifier designed to label cells as 'cancerous' or 'cancer-free'. Such a classifier would first examine a large number of both cancerous and cancer-free cells, in order to develop an accurate model. Then, using this model, the classifier could examine unlabelled cells and determine if they are cancerous. Furthermore, new labelled cells could be added to expand and/or clarify the model.

Unsupervised methods interpret data without requiring (or being able to make use of) existing labels on the data. DBSCAN is an example of an unsupervised clustering method. This method algorithmically finds connected, dense regions in a data set and labels them. DBSCAN does not require (and cannot use) prior labels on the data.

Semi-supervised methods make use of a small number of labelled objects, typically called *constraints*, in order to give direction to the algorithm and/or improve the result.

The majority of clustering algorithms are unsupervised: in fact, semi-supervised clustering is a relatively recent development. In their 1999 definition of clustering (referenced in Chapter 1), Jain et al. state that clustering is the “unsupervised classification of patterns (observations, data items, or feature vectors) into groups (clusters)” [Jain et al., 1999]. Semi-supervised (sometimes also known as constrained or constraint-based) clustering algorithms have been developed as a means to allow end users (ideally, domain experts in their given fields) to provide additional information to clustering algorithms (in the form of constraints). Consider

DBSCAN: this algorithm's two input parameters, *MinPts* and *Eps*, are difficult for end users to set (that is, even with a deep understanding of the data that is to be clustered, the correct values for these parameters is not immediately obvious). As well, the correct labels for a number of objects in the data set may be known by the end user, but DBSCAN is unable to take advantage of this information. A semi-supervised clustering algorithm will make use of this information. Grira et al. summarize the reasons for semi-supervised clustering when they state that "in many cases a small amount of knowledge is available concerning pairwise (must-link or cannot-link) constraints between data items or class labels for some items. Instead of simply using this knowledge for the external validation of the results of clustering, one can imagine letting it 'guide' or 'adjust' the clustering process, i.e. provide a limited form of supervision" [Grira et al., 2004].

Although a wide variety of different types of constraints exist, the majority of semi-supervised methods use only two types of constraints: *should-link* and *should-not-link*. A should-link constraint exists between two objects in the data set when those objects are assigned the same label by a user. Objects that share a should-link constraint should be placed in the same cluster by the algorithm. Similarly, when two objects are assigned different labels, a should-not-link constraint exists between them, and the two objects should not be placed in the same cluster. Providing should-link and/or should-not-link constraints to a semi-supervised clustering algorithm typically increases the probability that the objects in the constraint(s) will be placed in the correct cluster(s). It is worth noting that, for most algorithms, should-link constraints are considered transitive (that is, if a should-link constraint exists between *A* and *B*, and a should-link constraint exists between *B* and *C*, then a should-link constraint also exists between *A* and *C*). *Must-link* and *cannot-link* constraints are a stronger form of constraint – that is, a must-link constraint specifies that objects must be placed in the same cluster, while a cannot-link constraint specifies that objects cannot be placed in the same cluster. Some algorithms mistakenly claim to use must-link and/or cannot-link constraints, but allow these constraints to sometimes be violated: in these cases, it is more correct to state that the constraints being used are should-link and/or should-not-link.

COP-KMEANS is "a general method for incorporating background knowledge in the form of instance-level constraints into the k-means clustering algorithm" [Wagstaff et al., 2001]. It is one of the first semi-supervised clustering algorithms. Wagstaff et al. describe the traditional k-means clustering algorithm as an initial setup step of selecting *k* cluster centers, followed by an

iterative two-step process of assigning objects to their nearest cluster and then updating the cluster centers. COP-KMEANS modifies the k-means algorithm by making use of must-link and cannot-link constraints. During the step of assigning objects to clusters, the algorithm checks that must-link and cannot-link constraints are not violated. If a constraint is violated, then the violating object will instead by assigned to a different cluster. In cases where an object cannot be assigned to any cluster (there is always some constraint violation), COP-KMEANS fails (returns an empty set of clusters).

Basu et al. propose two semi-supervised clustering algorithms based on k-means: Seeded-KMeans and Constrained-KMeans [Basu et al., 2002]. In both algorithms, the cluster center (or seed) that is initially used for a given cluster is not randomly generated: instead, the mean average of all labelled points that belong to the cluster is used as the cluster seed. Following this initialization step, Seeded-KMeans finds clusters using the traditional k-means algorithm – this means that points which have been labelled by the user may be re-labelled by the algorithm. Constrained-KMeans, on the other hand, does not allow the cluster membership of labelled points to be changed. Basu et al. recommend using Seeded-KMeans when the initial labelling may be noisy (imperfect), and using Constrained-KMeans otherwise.

The Constrained Complete-Link (CCL) algorithm is presented by Klein et al. This algorithm attempts "to take feature-space proximities, along with a sparse collection of pairwise constraints, and cluster in a space which is generally like the feature-space, but altered to accommodate the constraints" [Klein et al., 2002]. CCL modifies the feature space by treating the data set as a graph – points are vertices, and distances between points become weighted edges. First, distances between pairs of points in must-link constraints are set to 0. Next, an all-pairs-shortest-paths algorithm is run, in order to update the distances between any pairs of points that are now much closer due to the must-link constraints. Klein et al. note that incorporating cannot-link constraints in a similar manner is an NP-complete problem. Therefore, they simply choose to set the distances between pairs of points in cannot-link constraints to the maximum current distance between any pairs of points plus 1, without updating other distances. These changes result in a modified similarity matrix for the data set. As a final step, unmodified complete-link hierarchical agglomerative clustering is run on this modified similarity matrix.

Xing et al. present "an algorithm that, given examples of similar pairs of points in $\mathbb{R}^n$, learns a distance metric that respects these relationships" [Xing et al., 2002]. The main

component of this distance metric is a positive semi-definite matrix. Xing et al. provide an iterative algorithm which produces a matrix that minimizes the squared distance between pairs of points. Additional requirements are added to ensure that the distance metric produced does not transform the distances between all points to 0 (which would trivially solve the minimization problem). Essentially, Xing et al.'s algorithm takes a data set and a set of must-link constraints and produces a distance metric which transforms the set of attributes such that objects in the must-link constraints are brought closer together. Although Xing et al.'s method is not a clustering algorithm, it is semi-supervised, and can be used to modify any clustering algorithm.

C-DBSCAN is a density-based, constraint-based algorithm that makes use of DBSCAN [Ruiz et al., 2007]. This algorithm first partitions the data set into a tree of axis-parallel hypercubes, with leaf nodes in this tree containing at least *MinPts* points. Next, DBSCAN is run on each leaf node (with particular values for *MinPts* and *Eps*), which will result in clusters in dense leaves and noise points (outliers) in sparse leaves. Ruiz et al. note that special action is taken if two points in a cannot-link constraint both appear in the same leaf node: in this case, DBSCAN is not run on the leaf, and instead all points in the leaf are temporarily assigned to be singleton clusters. Next, pairs of clusters in separate leaf nodes which share a must-link constraint are merged. Finally, an iterative step of merging the two most closely density-reachable clusters is repeated, until no more mergers are possible. Note that cannot-link constraints are enforced during this final merger step: that is, two clusters will not be merged if that merger would violate a cannot-link constraint; instead, the next two most closely density-reachable clusters are examined.

Böhm and Plant propose HISSCLU, "a hierarchical density-based approach to semi-supervised clustering which avoids the use of explicit constraints" [Böhm & Plant, 2008]. Rather than making use of must-link or cannot-link constraints, HISSCLU uses labelled objects as starting points to expand clusters from. Simultaneously, each density-based cluster is expanded by adding the most strongly density-connected unlabelled object to the cluster. In this manner, each unlabelled object will eventually be added to a cluster. Böhm and Plant note that border lines between different clusters can sometimes appear relatively random – in particular when the two clusters should actually be one, but were not labelled as such. In order to solve this problem, Böhm and Plant introduce a local label-based weighting function. Prior to the clustering step, this function transforms distances between points based on the distances to labelled objects. The

final output of HISSCLU is a cluster diagram which is similar to the reachability plot produced by OPTICS. This cluster diagram shows the density level at which various objects were added to their respective clusters – and, like a reachability plot, clusters are seen as 'valleys'.

SSDBSCAN is a "semi-supervised density-based clustering algorithm that takes advantage of background knowledge in the form of labeled instances to extract the intrinsic density of density-based clusters in a dataset" [Lelis & Sander, 2009]. This algorithm modifies DBSCAN by using cannot-link constraints to give each density-based cluster its own density threshold (rather than having a single, global threshold). Like HISSCLU, SSDBSCAN expands density-based clusters starting from labelled objects. Each cluster is expanded until an object with a label different from the original object is encountered. At this point, the least dense link between any two objects in the cluster is found, and the density threshold of the cluster is set to be just greater than the density of this link. This threshold causes the newly encountered labelled object (as well as potentially other objects) to be separated from the cluster. All of the objects that remain density-connected are saved as a cluster, and the process repeats for the next labelled object that has not yet been assigned to a cluster.

It is important to note that HDBSCAN* [Campello et al., 2013, Campello et al., 2015] (which was described in the previous section) is another example of a semi-supervised density-based clustering algorithm.

Wagstaff et al. make note of the fact that the quality of clustering produced by a semi-supervised method can vary greatly, depending largely on the quality of the constraints provided by the user [Wagstaff et al., 2006]. In fact, in some cases, the partition produced by a semi-supervised method may be worse than the partition produced by an unsupervised method. Wagstaff et al. define two properties to measure the quality of a set of constraints: *informativeness* and *coherence*. Constraints are considered informative if they cause a change in the algorithm's output such that the constraint becomes satisfied. If an algorithm would have already satisfied the constraint without requiring it as input, than that constraint is not informative for that algorithm. A set of constraints is considered coherent if the constraints are in agreement with each other when considering a given distance metric on the data set.

There are several drawbacks shared by the semi-supervised methods that have been mentioned. Firstly, for many of the algorithms, the user is forced to set several parameters in addition to providing labelled objects. These parameters may be hard to set, and in many cases,

the semi-supervised version of an algorithm requires as many or more parameters than the unsupervised version (SSDBSCAN is a notable exception). Secondly, many semi-supervised clustering algorithms make poor use of cannot-link constraints, or use cannot-link constraints in a manner which does not logically follow from their use of must-link constraints (CCL and C-DBSCAN both suffer from these issues). Finally, as was the case with the density-based algorithms examined, none of these semi-supervised algorithms are able to handle high-dimensional data sets: that is, they cannot detect clusters in subsets of the full set of attributes.

2.4 Subspace and Projected Clustering

As the number of attributes (dimensions) in a data set increases, the distances between objects become more and more similar [Aggarwal et al., 2001, Beyer et al., 1999, Hinneburg et al., 2000]. As previously stated in Chapter 1, this effect is known as the 'curse of dimensionality'. Since the vast majority of clustering algorithms rely heavily on distance as a means to determine whether objects are similar or dissimilar, most traditional clustering algorithms fail to produce useful results on high dimensional data sets. The 'curse of dimensionality' has prompted the creation of two new general paradigms for clustering: subspace and projected. Both subspace and projected clustering algorithms search for clusters in subsets of the full set of attributes; however, they differ in their methodology and their definition of what constitutes a cluster. One important concept that both clustering paradigms share is the notion of a *relevant* subspace. Generally speaking, a relevant subspace for a given cluster consists only of relevant attributes – that is, attributes which help differentiate the cluster (improve the quality of the cluster and make it easier to detect). Removing relevant attributes from a subspace or adding irrelevant attributes to a subspace should make the cluster more difficult to differentiate. The best (most relevant) subspace for a given cluster is the subspace which consists of all the relevant attributes, and none of the irrelevant attributes.

Before examining subspace and projected clustering methods, it is important to note that other possible solutions to the 'curse of dimensionality' exist, such as dimensionality reduction techniques. Agrawal et al. explain that these techniques “transform the original data space into a lower dimensional space by forming dimensions that are linear combinations of given attributes” [Agrawal et al., 1998]. Additionally, users may manually remove certain attributes from a data set before clustering. Unfortunately, both of these solutions have several shortcomings. Firstly,

clusters may exist in different subspaces (having different sets of relevant attributes), and so globally removing or normalizing a dimension may result in a loss of information, or even make a cluster impossible to detect. Secondly, although many attributes may be noisy (contribute no meaningful information to the data set), it can be difficult for a user to determine which dimensions these are, and dimensionality reduction techniques may consider these attributes just as important as other, truly meaningful attributes. Finally, the new set of attributes produced by dimensionality reduction techniques may be "difficult to interpret, making it hard to understand clusters in relation to the original data space" [Agrawal et al., 1998]. Ideally, subspace and projected clustering algorithms are able to find clusters in subspaces of the original set of attributes, even with the presence of noisy attributes.

In their survey of subspace and projected clustering techniques, Moise et al. state that "projected clustering techniques define a projected cluster as a pair (X, Y), where X is a subset of data points, and Y is a subset of data attributes, so that the points in X are 'close' when projected onto the attributes in Y, but they are 'not close' when projected onto the remaining attributes" [Moise et al., 2009]. Moise et al. go on to note that projected clustering algorithms usually require (and rely heavily upon) several user-defined parameters (such as the desired number of clusters), usually only allow an individual object to have membership in a single cluster, and usually have difficulty finding clusters with a small number of attributes when the data set has a large number of attributes. Perhaps the most commonly shared similarity among projected clustering methods is the manner in which clusters are found. Typically, projected clustering algorithms start in either the full dimensional space or in some lower-dimensional projections and use some heuristics to find initial 'tentative clusters'. These 'tentative clusters' may start to be constructed with attributes only, or with objects only, or sometimes both. Some process (usually iterative) is then used to find and/or refine the attributes and/or objects. A some point this process stops, and a final partition is returned. Projected clustering algorithms usually rely heavily on the quality of the initial 'tentative clusters' for the end result.

PROCLUS (PROjected CLUstering) is the first projected clustering algorithm, and it "returns a *partition* of the data points into clusters, together with the sets of dimensions on which points in each cluster are correlated" [Aggarwal et al., 1999]. Aggarwal et al. note that PROCLUS has two input parameters (k, the number of clusters, and l, the average number of dimensions per cluster) and three phases: an initialization phase, an iterative phase, and a

refinement phase. In the initialization phase, PROCLUS randomly selects a number of points from the data set, and then greedily selects a subset of these points to act as potential cluster medoids. This subset is selected by choosing points from the random set that are far apart from each other in the full set of dimensions – these points are more likely to be members of different clusters than points which are very close to each other. Next, in the iterative phase, PROCLUS constructs clusters from a sample of the medoids, and then continually replaces the 'worst' cluster and corresponding medoid in an attempt to find a 'better' set of clusters. Dimensions for each medoid (and subsequent cluster) are chosen by examining points near the medoid in the full set of dimensions, and then selecting the dimensions along which these points are nearest to the medoid. Points are then assigned to their nearest medoid, with distances calculated only in the set of dimensions relevant to each respective medoid. A score for the current partition is calculated by averaging the distances between points and their cluster medoids (again, considering only the relevant set of dimensions for each cluster). Clusters with few points in them are considered 'bad', and the corresponding medoids are replaced, which causes new dimensions and clusters to be calculated. This iterative process repeats until the 'best' partition found so far (determined by score) cannot be improved upon. Finally, in the refinement phase, the set of dimensions for each cluster are re-calculated by examining the current set of points for the cluster. Points are then re-assigned to the appropriate cluster medoids, and points which are far away from every medoid are labelled as outliers.

Yip et al. present HARP, a “Hierarchical approach with Automatic Relevant dimension selection for Projected clustering” [Yip et al., 2004]. HARP determines if an attribute is relevant for a given cluster by comparing the variance of cluster members on the attribute to the global variance of all objects on the attribute. This hierarchical algorithm begins with each cluster as a separate object, and then gradually merges clusters. A 'merge score' is computed for each potential cluster merger, and the merger with the highest score is performed at each step. This score is high if two clusters share many relevant attributes. Two thresholds, d_{min} (the minimum number of dimensions per cluster) and R_{min} (the minimum relevance per attribute) are used to guide this process. These two thresholds start at the most strict possible values (meaning that only identical objects are merged), and then are gradually relaxed as the algorithm progresses. HARP returns when either d_{min} and R_{min} reach some minimum values, or when a user-defined number of clusters is found. As HARP reaches the end of its execution, a two-phase outlier

handling mechanism is performed.

DOC (Density-based Optimal projective Clustering) is a "Monte Carlo algorithm for iteratively computing projective clusters" [Procopiuc et al., 2002]. Procopiuc et al. define a projective cluster as a set of points in a subspace (set of dimensions) such that number of points in the cluster is a least a fraction α of the number points in the data set, and all clustered points in the subspace are within a hypercube of width w. Furthermore, the authors define an objective function $\mu(a,b) = a(1/\beta)^b$ to measure the quality of a cluster, where a is the number of points in the cluster and b is the number of attributes. The variables α, w, and β are all user-defined. DOC repeatedly randomly selects a single point p from the data set, along with a 'discriminating set' X of randomly chosen points. All dimensions in which the distances between p and all points in X are less than w are selected as relevant for the cluster; then, the set of points within the hypercube is determined. A score for any valid cluster constructed in this manner is computed (using the objective function $\mu(a,b)$), and the best cluster found after a certain number of iterations is returned. Procopiuc et al. note that DOC can be quite inefficient, and so they introduce FASTDOC, a potentially less accurate but much faster version of the algorithm. FASTDOC's most significant change is that the complete cluster is not computed during each iteration: instead, only the set of dimensions is stored. After the final iteration, FASTDOC selects the largest set of dimensions found and expands this set into a cluster.

Moise et al. present P3C (Projected Clustering via Cluster Cores), "a robust algorithm for projected clustering that can effectively discover projected clusters in the data while minimizing the number of parameters required as input" [Moise et al., 2006, Moise et al., 2008]. P3C partitions each attribute in the data set into a number of bins, and then performs a goodness-of-fit test to determine which attributes have a non-uniform distribution of objects. The densest bin in such an attribute is marked, and then this process is repeated on the remaining bins in the attribute until the remaining un-marked bins display a uniform distribution. Next, adjacent bins are merged into 'intervals'. Intervals from different attributes are combined into 'cluster cores' if all of these intervals share a number of objects in common which is higher than expected. A Poisson distribution is used to determine the likelihood of seeing a given number of objects in a given interval. A probabilistic membership matrix is then generated according to which objects appear in which cluster cores. This probabilistic membership matrix is used to initialize the Expectation Maximization (EM) clustering algorithm. EM clustering is then run in a subspace

which consists only of attributes that were originally marked non-uniform. The probabilistic memberships assigned by EM clustering are then converted into hard memberships, outliers are labelled, and relevant attributes are assigned to each cluster by testing the uniformity of cluster members on attributes.

Moise and Sander propose a “novel problem formulation that ensures that found subspace clusters actually stand out in the data in a statistical sense”, as well as “an approximation algorithm STATPC for the problem” [Moise & Sander, 2008]. The authors describe a hyper-rectangle *H* in a given subset of attributes as being 'statistically significant' if it contains significantly more points than should be expected (a uniform distribution of points is assumed). The significance threshold for 'statistical significance' is set to some user-defined value. A hyper-rectangle *H* is considered 'explained' by another set of hyper-rectangles if, given the distribution of points in these hyper-rectangles and the distribution of noise points, the number of points in *H* is not significantly lower or higher than expected. Moise and Sander describe a projected clustering solution (partition) for a data set as a minimal set of statistically significant hyper-rectangles (clusters) such that this set explains all other statistically significant hyper-rectangles in the data set. STATPC builds a candidate set of subspace cluster iteratively. First, a point is randomly selected from among all points that have not yet been clustered. Next, 2-dimensional rectangles are built around the selected point, and the rectangles are ranked by the number of points inside of them. Attributes which appear in highly ranked rectangles significantly more often than expected are used to construct an initial subspace. This subspace is expanded by finding other attributes in which the points in the rectangle are not uniformly distributed – this process is repeated (finding new attributes and then updating the set of points) until no new attributes can be added. Finally, a number of statistically significant minimum bounding hyper-rectangles are built around the randomly selected point, and the hyper-rectangle that best explains the other newly-built hyper-rectangles is saved as a candidate subspace cluster. This process is repeated until all points have been placed in a candidate cluster. In a final step, STATPC greedily selects clusters from the candidate set which explain as many other candidate clusters a possible, until all of the candidate clusters have been explained. This final set of clusters is then returned.

In contrast to projected clustering methods, subspace methods are always density-based, and find all valid clusters in all possible subspaces of a data set. Moise et al. note that subspace clustering algorithms “use global density thresholds for detecting subspace clusters in subspaces

of increasing dimensionality. The global density thresholds guarantee some anti-monotonic properties that are used to avoid an exhaustive search though all possible subspaces" [Moise et al., 2009]. Unlike most projected clustering algorithms, subspace algorithms allow each object membership in multiple clusters. Subspace clustering algorithms typically discover clusters in a bottom-up approach, finding dense regions in low dimensionality and then merging these regions in higher dimensionality. All clusters found in the subspaces of increasing dimensionality are returned.

CLIQUE is a grid-based subspace clustering algorithm presented by Agrawal et al. The authors state that CLIQUE has "the ability to find clusters embedded in subspaces of high dimensional data, scalability, end-user comprehensibility of the results, non-presumption of any canonical data distribution, and insensitivity to the order of input records" [Agrawal et al., 1998]. Subspace clusters are defined as connected dense units, where 'dense' means that each unit contains a least some user-defined percentage of the total number of points in the data set, and 'units' are non-overlapping multi-dimensional grid cells. CLIQUE starts with dense one-dimensional cells and merges them in an attempt to find dense two-dimensional cells. This process is repeated for increasingly higher dimensionality (note that non-dense cells are removed as candidates, due to anti-monotonic properties). Heuristics are used to prune the number of cells that must be searched – the authors note that this pruning increases efficiency, but may cause certain cells to be missed. Finally, connected dense units are joined together as clusters, and minimal DNF (disjunctive normal formal) descriptions for these clusters are then returned.

Kailing et al. note that in their algorithm SUBCLU (density-based SUBspace CLUstering), "the monotonicity of density-connectivity is used to efficiently prune subspaces in the process of generating all clusters in a bottom up way" [Kailing et al., 2004]. SUBCLU extends the original definition of density-based clustering presented in DBSCAN to subspaces: a subspace cluster is a maximal set of density-connected objects in a subspace. This algorithm finds all density-based clusters in all subspaces, starting with one-dimensional subspaces. Pairs of k-dimensional subspaces are combined into $k+1$-dimensional subspaces, with the anti-monotonic properties of density-connected sets used to prune subspaces – that is, if a particular k-dimensional subspace has no density-connected sets of points, then neither can any $k+1$-dimensional (or even higher dimensional) subspaces which contain the k-dimensional subspace. The DBSCAN parameters *Eps* and *MinPts* are used in SUBCLU are used to find core points and

clusters in subspaces.

DUSC (Dimensionality Unbiased Subspace Clustering) is a subspace clustering technique designed to take dimensionality bias into account. The authors define dimensionality bias as "a dependency of density on the dimensionality of the subspace: as dimensionality increases, average distances between objects increase and cluster radii grow" [Assent et al., 2007]. Assent et al. note that the expected density for any given subspace can be calculated, and that normalizing an object's measured density by this expected density gives an unbiased density measure: this measure can be compared across different subspaces. Furthermore, an object is considered dense if its density is *F* times greater than the expected density, where *F* is a user-defined parameter. Assent et al. note that, in high dimensions, even lone objects may be considered dense, and so they introduce another measure that prevents this from occurring. A subspace cluster is defined as a connected set of at least some minimum number of objects that are at least *F* times greater than expected. Additionally, clusters must not be redundant: that is, a significant percentage of a cluster's objects must not appear in another cluster of higher dimensionality. DUSC makes use of a depth-first approach to find subspace clusters. Redundant sets of objects, sets that are too small (do not contain enough points), and objects that cannot be dense in super subspaces of a given subspace are all pruned. It should be noted that DUSC does not define a core object using DBSCAN's *Eps-neighborhood*, rather an Epanechnikov kernel is used.

Like the density-based methods and the semi-supervised methods that were examined, the projected and subspace algorithms discussed have significant shortcomings. First of all, because many of the projected clustering algorithms examined do a significant portion of their distance calculations in the full dimensional space, they can easily miss low-dimensional subspace clusters. Secondly, many of the algorithms examined are quite slow – projected methods can perform a large number of iterations, and subspace methods can search an exorbitant number of subspaces, even when pruning a large percentage of the search space. Thirdly, because they use a global density threshold, subspace methods are often forced to either return a high number of mostly-redundant, largely-useless low-dimensional clusters, or else risk missing many high-dimensional clusters. This can make the output of a subspace method very difficult to interpret. Fourthly, many of these algorithms (both projected and subspace) introduce additional input parameters when compared to their full-space counterparts, making them even more difficult to

use. Finally, none of these algorithms are able to make use of constraints in any way.

2.5 Semi-Supervised Projected Clustering and Related Methods

An examination of existing literature would seem to indicate that very few semi-supervised projected clustering algorithms exist, compared to the number of semi-supervised clustering algorithms and/or the number of projected clustering algorithms. Additionally, as far as we can tell, no semi-supervised density-based projected clustering algorithms exist at the time of the publication of this thesis (although at least one semi-supervised density-based subspace clustering algorithm does exist). In this section, we will briefly examine several existing semi-supervised projected clustering methods, as well as a few related methods.

Yip et al. consider SSPC (Semi-Supervised Projected Clustering) to be the first semi-supervised projected clustering algorithm [Yip et al., 2005, Yip et al., 2009]. Constraints in the form of labelled objects and labelled dimensions can be provided to SSPC. SSPC is a k-medoid like algorithm that attempts to iteratively maximize an objective function. This objective function takes into consideration the within-cluster dispersion of each cluster and the number of attributes assigned (that is, declared relevant) for each cluster. SSPC starts by generating a number of 'seed groups'. Yip et al. state that "Each seed group contains a set of seeds that are expected to come from a single real cluster, and a set of dimensions estimated from the seeds to be relevant to the cluster" [Yip et al., 2005]. Several multi-dimensional grids (using different sets of dimensions) are constructed, and cells with a large number of objects in them (that is, dense cells) are selected. Objects in these dense cells (along with the corresponding dimensions) are then selected for seed groups. 'Private' seed groups are generated for clusters for which constraints have been provided, and each of these seed groups will only be used by their corresponding cluster. For these private seed groups, labelled objects influence which cells are selected, while labelled dimensions influence which dimensions are used when building the grids. A large number of 'public' seed groups are generated for the remaining clusters. For these public seed groups, seeds are selected that are well-separated from the already selected seeds.

After seed groups have been constructed, SSPC begins an iterative process. The first step of this process is to assign all objects in that data set to clusters (or noise). Objects are assigned to nearby cluster representatives, which are selected from the appropriate seed groups. Next, a

set of dimensions are assigned to each cluster in a manner that attempts to maximize the objective function. After this, the objective score of the current set of clusters is compared to the objective score of the best set of clusters found so far. If the current set is better, it replaces the best set. Finally, SSPC attempts to identify a 'bad cluster' and replace it's cluster representative. This entire process is repeated until either a given number of iterations is reached, or until no there are no improvements for a fixed number of iterations.

SSPC requires a number of input parameters. Notably, like all other k-medoid methods, SSPC requires *K*, the number of clusters. Additionally, SSPC requires a threshold value to determine which dimensions to select when assigning dimensions to clusters. This threshold value is compared to the variance of the cluster members on each dimension. The authors propose two alternatives: $m \in [0,1]$ and $p \in [0,1]$. The *m* parameter compares cluster variance on a given dimension to the variance of all objects on that dimension. If the probability density function of the data is known, the *p* parameter can be used to determine the probability that a given dimension is relevant for a given cluster. Note that SSPC has a number of other input parameters, such as the maximum number of iterations, the number of iterations without improvement before stopping, the dimensionality of the grids, etc.

Zhang et al. present S^3C (semi-supervised subspace clustering) [Zhang et al., 2011]. Note that, despite the name, we would argue their method is a projected one. S^3C consists of two phases. In the 'subspace forming phase', sets of constraints are combined and a subspace relevant to each respective combined constraint set is formed. In the 'data assignment phase', points are assigned to nearby clusters. The first step of the subspace forming phase is 'constraint-dimension correlating'. In this step, neighbors of pairs of points in constraints are compared. A dimension is considered relevant for a must-link constraint if the points in the must-link constraint share many neighbors on that dimension. Similarly, a dimension is considered relevant for a cannot-link constraint if the points in the cannot-link constraint share very few neighbors on the dimension. An input parameter is used to set a threshold for determining relevancy. In the 'constraints combining' step, sets of constraints that share a large number of dimensions are combined into 'constraint unions' using a transitive bottom-up process. Each of these unions will have 'backbone dimensions', which are relevant for every constraint in the constraint union, 'uncertain dimensions', which are relevant for some of the constraints, and unrelated dimensions. In the 'dimension selecting step', uncertain dimensions are either selected for their respective

constraint unions or rejected, based on whether these dimensions improve the consistency of the constraints (that is, based on how these dimensions change the average distances between points in must-link and cannot-link constraints). In the data assignment phase, points are assigned to nearby cluster centroids, finalizing the clusters.

VINAYAKA is a semi-supervised projected clustering algorithm that makes use of differential evolution [Gajawafa & Toshniwal, 2012]. Differential evolution methods iteratively attempt candidate solutions to a problem, and eventually terminate once a stopping criteria is met. VINAYAKA measures the fitness (quality) of potential solutions using a combination of an internal validation criteria known as SCQE (Subspace Clustering Quality Estimate) and an external validation criteria known as the Gini gain index. In its iterative phase, VINAYAKA goes through several steps of identifying points and attributes which are relevant to clusters (and then assigning these points and attributes). Gajawafa and Toshniwal provide very few details of the inner workings of the algorithm.

Guerra et al. present SesProC, a semi-supervised projected clustering algorithm that uses model-based clustering [Guerra et al., 2014]. SeSProC uses the EM (Expectation-Maximization) algorithm. Guerra et al. adapt an EM model to account for both labelled instances (constraints) and subspaces. Labelled instances are "used to initialize the known groups and as a starting point for selecting the final number of clusters" [Guerra et al., 2014]. Using an iterative process, SeSProc is able to find clusters in the data set for which labelled instances are not provided. This iterative process starts with the known (labelled) clusters, and at each step attempts to add one more cluster to the model. This new cluster is initialized using instances that do no fit the model well. If adding any cluster would lower the quality of the result (as determined by the model), SeSProc stops iterating and returns the current set of clusters, instead of adding this new cluster. SeSProc takes a single input parameter, *Cth* (candidate threshold), which specifies how many instances to initialize new clusters with when iteratively trying to add clusters.

Fromont et al. present a framework that encapsulates several existing subspace clustering methods, and also introduce a semi-supervised density-based subspace clustering algorithm known as SC-MINER [Fromont et al., 2009]. SC-MINER is a bottom-up algorithm that divides each dimension into a number of 'bins'. Lower dimensional dense bins are recursively combined into higher dimensional dense bins (subspaces). The monotonic properties of density allow large numbers of potential higher dimensional bins (subspaces) to be pruned from the search space.

Additionally, SC-MINER makes use of the monotonic and anti-monotonic properties of constraints to perform additional pruning, thereby further improving efficiency. Unlike projected clustering methods, SC-MINER outputs all of the higher dimensional clusters (objects and bins) that it finds, thereby potentially providing the user with far too many results to meaningfully understand.

A number of algorithms exist that perform semi-supervised dimensionality reduction (feature selection) as a part of the clustering process. While these methods are related to semi-supervised projected clustering, they typically reduce the set of attributes globally, rather than on a per-cluster basis. As well, these methods may transform the set of attributes, or construct a new, smaller set of attributes that represent the larger, original set of attributes. These transformed attributes can be difficult to relate to the original set of attributes (thereby making the final result difficult to interpret).

Handl and Knowles present a semi-supervised feature selection algorithm that makes use of multiobjective optimization [Handl & Knowles, 2006]. Their algorithm uses an optimizer known as PESA-2 to iteratively produce candidate subspaces. K-means clustering is performed on each of these subspaces, and the resulting clusters are evaluated using internal and external evaluation techniques. The algorithm performs a fixed number of iterations through optimization, clustering, and evaluation, and selects best set of features through all iterations based on objective functions.

Yan and Domeniconi explore using a semi-supervised ensemble method to learn new distances between objects in a data set [Yan & Domeniconi, 2006]. They construct a number of random subspaces and, making use of must-link and cannot-link constraints, learn a new distance matrix in each subspace. K-means clustering is then performed in each of these subspaces using their respective distance matrices. The ensemble of clusterings is then combined into a bipartite graph (which effectively re-defines the distances between objects globally). Finally, spectral graph partitioning is performed on this graph to produce a final clustering result.

SCREEN (Semi-supervised Clustering method base on spheRical K-mEans via fEature projectioN) is a semi-supervised clustering method that uses global feature projection [Tang et al., 2007]. In a pre-processing step, SCREEN uses constraints to combine objects involved in must-link constraints. SCREEN then uses the constraints to guide feature projection, replacing the original set of dimensions with a modified lower dimensional subspace. Constrained

Spherical K-means clustering (a semi-supervised clustering method) is then performed on this lower dimensional subspace. In a final post-processing step, SCREEN transforms the combined objects back into points found in the original data set.

Zhang et al. present SSDR (semi-supervised dimensionality reduction) [Zhang et al., 2007]. SSDR computes a transformation matrix for projecting high dimensional data into a lower dimensional space. This algorithm takes into account must-link constraints, cannot-link constraints, and the variance of unlabelled points as well. Zhang et al. do not integrate any clustering methods directly into SSDR.

Constrained Locality Preserving Projections (CLPP) is presented by Cevikalp et al. [Cevikalp et al., 2008]. This method modifies Locality Preserving Projections (LPP), which uses an adjacency graph to perform dimensionality reduction. Edge weights between points are assigned based on neighborhoods, with must-link and cannot-link constraints used to modify these edge weights. Additionally, points which are neighbors to constrained points also have their edge weights modified. CLPP produces a transformation matrix which modifies the original data set. Cevikalp et al. do not integrate any clustering methods directly into CLPP.

Although all of the algorithms mentioned in this section are semi-supervised and provide some solution to the curse of dimensionality, none of them (apart from SC-MINER) are density-based. SC-MINER is a subspace clustering algorithm, which has the disadvantage of producing a very large number of mostly-redundant low-dimensional clusters. Additionally, note that many of the dimensionality reduction techniques mentioned modify the attributes in the data set, making the results difficult to interpret.

Chapter 3: Background on Density-Based Clustering

In this Chapter, we introduce the basic concepts of density-based clustering, and then examine HDBSCAN* and FOSC. This Chapter begins with a detailed explanation of density-based clustering definitions (including concepts such as density-connectivity, core distance, mutual reachability, and minimum-eps-connectivity). Next, we explain how HDBSCAN* produces a density hierarchy (also called a dendrogram) and a corresponding cluster tree, and how FOSC can be used to extract a flat cluster from the dendrogram and cluster tree. HDBSCAN* uses a measure known as cluster stability [Campello et al., 2013b] and Campello et al., 2015] in order to extract clusters from the HDBSCAN* hierarchy. Finally, we briefly explain how the FOSC framework can be used to leverage constraints when extracting a flat cluster partition.

3.1 Density-Based Definitions

Most of the definitions of density-based clustering that follow are taken from the HDBSCAN* papers [Campello et al., 2013b, Campello et al., 2015]. These definitions can be considered extensions and modifications of definitions that were originally provided in the explanation of DBSCAN [Ester et al., 1996], which are in turn extensions of theoretical definitions provided by Hartigan [Hartigan, 1975]. Note that we adjust some of the terminology slightly, as well as introduce several definitions of our own.

Consider a data set of n objects $o_1, o_2, \ldots o_n$ each of which has d attributes. The kth attribute of object o_z can be denoted as o_{zk}. We consider only data sets where attributes are values in the range of real numbers, and where each object is not missing any attributes (that is, each object o_z has a complete set of d attributes). Given that each object consists of d real numbers, objects can be represented as points in d-dimensional space, and the distances between objects can be calculated. A variety of different distance functions (such as Manhattan, Euclidean, or Supremum) can be used to compute these distances. We use the Euclidean distance function to compute the distances between objects in our algorithm. It is worth noting that other metric distance functions can easily be substituted in, and may change the clustering result.

There are a variety of different ways to compute an object's density. A large number of different density kernels (such as uniform, triangular, Epanechnikov, and Guassian) exist, each of which has a different method for determining object density. In general, for any given density kernel, an object o_z will be considered *dense* (also known as a *core object*) if there are a large number of other objects very close to o_z. On the other hand, if o_z is well separated from other objects in the data set, (that is, there are few or no other objects near it) o_z will not be considered dense. DBSCAN uses a square density kernel. For such a kernel, an object o_z is dense if at least *MinPts* objects (including o_z itself) are within a distance *Eps* of o_z (both *Eps* and *MinPts* are parameters set by the user). All of the objects within a distance *Eps* of o_z are part of o_z's *Eps-neighborhood*.

Two objects o_y and o_z are *directly-density-connected* if they are both dense (that is, they are both core objects) and within a distance of *Eps* of each other (that is, both objects are in the other's Eps-neighborhood). This is a symmetric relationship (if o_y is directly-density-connected to o_z, then o_z must also be directly-density-connected to o_y). By definition, objects that are not dense cannot be directly-density-connected to any other objects. Two objects o_y and o_z are *density-connected* if they are either directly-density-connected, or if there is a 'chain' of directly-density-connected objects o_y, o_1, o_2, … o_z such that each object in the chain is directly-density-connected to the next object in the chain. Like direct-density-connectivity, density-connectivity is a symmetric relationship, and non-dense objects cannot be density-connected to any other objects. A *density-based cluster* is a maximal non-empty set of objects such that all objects in the set are density-connected to each other. Suppose there is some object o_y that is a member of cluster c_a. If o_y is density-connected to some object o_z, then o_z must also be a member of cluster c_a, since c_a is maximal by definition.

HDBSCAN* expands on DBSCAN's definition of density by using *k-nearest-neighbors* distance, or *knn* distance. The *core distance* of an object o_z is the distance from o_z to whichever object is the *MinPts* closest object to o_z (including o_z itself) – in other words, this is the distance to o_z's *MinPts*-nearest-neighbor. Given some value for *MinPts*, the core distance of object o_z is the minimum setting for *Eps* such that o_z will be considered dense – that is, if *Eps* is set to a value equal to or greater than o_z's core distance, o_z will be a core point. We use $CoreDist_{MinPts}(o_z)$ to denote an object o_z's core distance, given a particular setting of *MinPts*. See Figure 1 for a visual representation of core distance and Eps-neighborhood. The *mutual reachability distance* of two

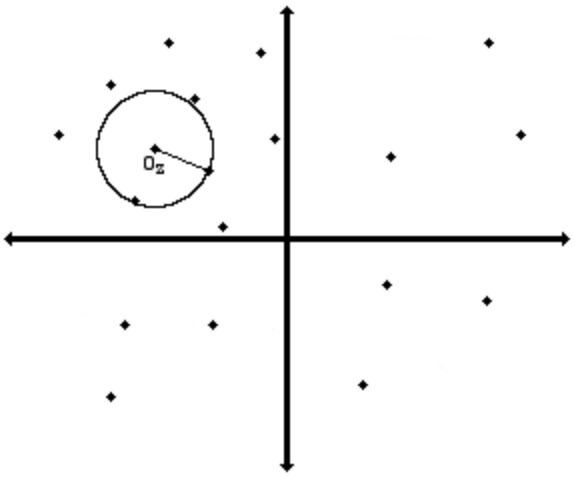

Fig 1: A 2-dimensional example data set. The core distance and *Eps*-neighborhood for object o_z when *MinPts* = 3 are displayed.

objects o_y and o_z is defined as $maximum(CoreDist_{MinPts}(o_y), CoreDist_{MinPts}(o_z), Dist(o_y, o_z))$. Mutual reachability distance represents how 'close' two objects are in terms of density: that is, how strongly connected the objects are when both density and distance are considered. We use $MRD_{MinPts}(o_y,o_z)$ to denote the mutual reachability distance between two objects o_y and o_z, given a particular setting of *MinPts*. It is worth noting that when $MinPts \leq 2$, the mutual reachability distance of any two objects will simply be the distance between the objects.

A *mutual reachability graph* is a complete graph which represents all *n* objects in the data set as vertices. Weights are assigned to the edges equal to the mutual reachability distance between the respective objects represented by the vertices (note that these weights are, of course, dependent on *MinPts*). A mutual reachability graph is useful for finding all density-based clusters at a given *Eps* level: removing all edges with weight greater than *Eps* yields connected components that represent density-based clusters. Vertices with no edges for a given *Eps* level represent non-dense (noise) objects (provided $MinPts \geq 2$). See [Campello et al., 2013b] and [Campello et al., 2015] for a more in-depth explanation of mutual reachability graphs. For computational efficiency (and for simplicity's sake), it is important to note that a *minimum spanning tree*, rather than the complete mutual reachability graph, can be used to find density-based clusters at a given *Eps* level. As is the case with the mutual reachability graph, removing all edges with weight greater than *Eps* gives the density-based clusters at that *Eps* level. A minimum spanning tree is a connected graph with a minimal number of edges and minimal total combined weight across all edges, and can be constructed using Prim's algorithm [Prim, R. C.,

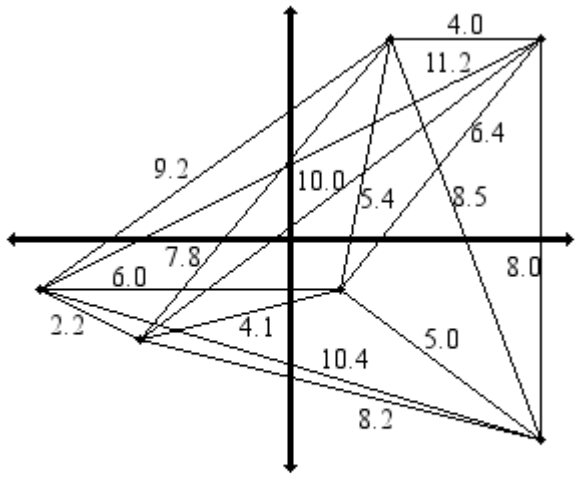

Fig 2a: A mutual reachability graph (*MinPts* = 2) of a 2-dimensional data set. Edge weights (mutual reachability distances) are displayed.

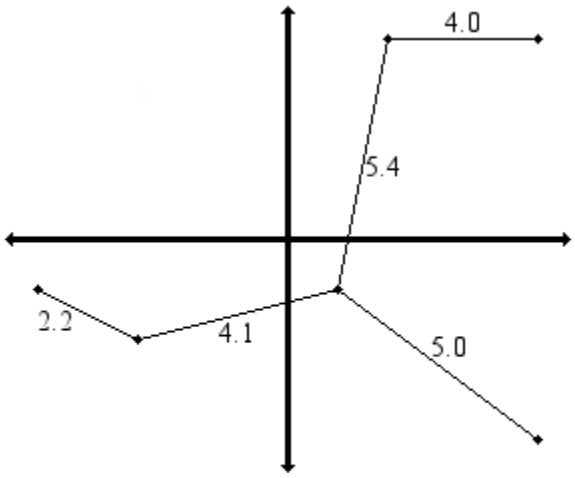

Fig 2b: A minimum spanning tree constructed from the mutual reachability graph seen in Figure 2a.

1957]. See [Campello et al., 2013b] and [Campello et al., 2015] for proof that a minimum spanning tree can be used in place of a mutual reachability graph. Figures 2a and 2b show an example of a mutual reachability graph and the minimum spanning tree constructed from the mutual reachability graph.

We define the *minimum-eps-connectivity* of two objects o_y and o_z as the maximal edge in the path connecting o_y and o_z in the minimum spanning tree of the mutual reachability graph. The lower the minimum-eps-connectivity of o_y and o_z, the more strongly density-connected the objects are (of course, the opposite also holds true). Minimum-eps-connectivity can also be defined as the minimum *Eps* value such that o_y and o_z are still members of the same density-based cluster. At lower *Eps* values, o_y and o_z will no longer be density-connected. We use $MEC_{MinPts}(o_y,o_z)$ to denote the minimum-eps-connectivity between two objects o_y and o_z, given a particular setting of *MinPts*.

3.2 HDBSCAN* and FOSC

Given a minimum spanning tree of a mutual reachability graph, a hierarchy of density-based clusters (a dendrogram) can be produced. This dendrogram is created by iteratively removing the edge with the largest weight from the minimum spanning tree – this is equivalent to iteratively decreasing the *Eps* level and finding the density-based clusters at each level. At the top level of this dendrogram is the root cluster, which contains all *n* objects in the data set. As the *Eps* level decreases, some objects will become non-dense (noise), and clusters will split into

multiple smaller clusters. At some level, all objects in the data set will be non-dense (noise). Note that if $MinPts \leq 1$, then all objects will have a core distance of 0, and no object will ever become non-dense (although clusters will still split).

HDBSCAN* uses such a minimum spanning tree to construct a dendrogram. The algorithm begins by calculating the core distance for each object in the data set. Next, the minimum spanning tree is constructed (note that for computational efficiency, the full mutual reachability graph is never constructed). HDBSCAN* then iteratively removes the edge(s) with the largest weight from the minimum spanning tree, tracking clusters and objects throughout the process. The output of this algorithm is a complete dendrogram which includes all *Eps* levels at which changes (objects becoming non-dense or clusters splitting) occur. It is important to note that HDBSCAN* uses two input parameters: *MinPts* and *MinClSize*. The *MinPts* parameter has already been discussed extensively. As mentioned in section 2.2, the *minClSize* parameter specifies the minimum number of density-connected core points that are required to form a cluster. With the introduction of *minClSize*, some core points may no longer be part of a cluster. A *cluster tree* can be constructed from the dendrogram, which is essentially a simplified version of the dendrogram that displays only the hierarchy of clusters, and omits details about individual objects. Obviously, the root cluster is at the root of this cluster tree. See [Campello et al., 2013b] and [Campello et al., 2015] for a more detailed explanation of HDBSCAN*.

HDBSCAN* employs FOSC (Framework for Optimal Selection of Clusters), "a framework for the optimal extraction of flat clusterings from local cuts through cluster hierarchies" [Campello et al., 2013a] in order to extract a flat partition from the cluster hierarchy (dendrogram). FOSC uses a bottom-up process to select the clusters with the best total score from the cluster tree (we discuss a method for scoring clusters in the next paragraph). It is important to note that either a parent cluster or its child clusters can be selected for the flat partition, but not both (since a single object cannot have multiple labels in a flat partition, and all objects which have membership in a child cluster also have membership in the parent cluster). For each cluster c_a in the cluster tree, beginning at the leaf clusters, the score of c_a is compared with the combined total score of the best set of descendant clusters found so far (for leaf clusters, there will be no best set of descendants, and for the parents of leaf clusters, the best set of descendants will simply be its children). Whichever group (c_a itself or the best set of descendants) has higher total score is then passed up the tree to c_a's parent. This process is

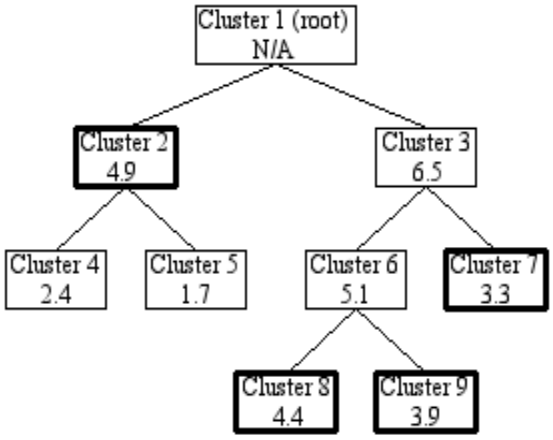

Fig 3: An example cluster tree. Stability values are displayed for each cluster. Clusters selected as the final flat partition (2, 7, 8, and 9) are bolded.

repeated until a final set of clusters with the highest possible total score is passed to the root cluster (depending on the scoring method used, the root cluster itself may or may not be eligible for selection). The final flat partition is constructed from this set of clusters, with corresponding labels assigned to the objects which are cluster members. See [Campello et al., 2013b] and [Campello et al., 2015] for a more in-depth explanation of this process. Figure 3 provides an example cluster tree.

Campello et al. [Campello et al., 2013a], [Campello et al., 2013b], [Campello et al., 2015] present a method for scoring each cluster in a cluster tree, called *Cluster Stability*. Roughly speaking, a cluster's stability is a measure of how long a cluster 'survives' within the dendrogram, as well as how many objects are part of the cluster. Given an object o_z that is a member of cluster c_a, o_z's contribution to c_a's cluster stability is equal to the difference between the density level at which o_z becomes a member of c_a and the density level at which o_z is no longer a member of c_a (note that a density level and an *Eps* level are the inverse of each other, as density increases when *Eps* decreases, and vice-versa). The stability contribution of o_z to c_a can be written as $d_{leave} - d_{join}$, where d_{leave} is the density level in the hierarchy at which o_z leaves c_a and d_{join} is the density level in the hierarchy at which o_z joins c_a. Since density levels and *Eps* levels are the inverse of each other, o_z's contribution to c_a's cluster stability can be written as $(1 / Eps_{leave}) - (1 / Eps_{join})$. Note that o_z may cease to be a member of c_a at some *Eps* level either because o_z becomes non-dense (noise) or because c_a splits into several smaller clusters. The total cluster stability of c_a is simply the sum of the stability contributions of each object that is a member of c_a. It is worth noting that

all objects which are members of c_a become members at the same *Eps* level (the level at which the cluster first begins to exist). Campello et al. choose to make the root cluster ineligible for selection as part of the flat partition when using cluster stability (although a cluster stability score can still easily be computed for the root cluster). If the root cluster were to be selected, all objects in the data set would be labelled as members of the same cluster (which is not a meaningful clustering result). See [Campello et al., 2013b] and [Campello et al., 2015] for more details on the cluster stability measure.

3.3 Applying Constraints

One aspect of FOSC that has not yet been discussed is making use of constraints provided by the user. By constraints, we mean pairs of objects that the user has either specified should be put in the same cluster (should-link) or should not be put in the same cluster (should-not-link). These types of constraints are also known as *pairwise constraints* (because each constraint involves only two objects). Of the several methods that Campello et al. introduce for integrating constraints into flat partition extraction, we choose to use a convex combination function in the form *adjusted cluster score* = α * *constraint satisfaction* + $(1 - \alpha)$ * *cluster score* [Campello et al., 2013a]. This function adjusts cluster scores (cluster stability in the case of HDBSCAN*) by taking into consideration how well clusters fulfill constraints. The α variable is a user-defined input parameter that must be in the range [0,1] and adjusts how heavily cluster scoring should be biased towards constraint satisfaction. Setting $\alpha = 1$ causes only constraint satisfaction to be taken into account for cluster scoring, while setting $\alpha = 0$ causes constraint satisfaction to be ignored entirely. We choose to use such a convex combination function because it allows the user to set how important constraint satisfaction is when determining which clusters to select for the flat partition.

One important element of this convex combination function is that both cluster score (cluster stability) and constraint satisfaction must be in the range [0,1]. Unfortunately, cluster stability values do not naturally fall into this range. Campello et al. describe a process for normalizing cluster stability values by dividing the cluster stability score assigned to each cluster by the sum total stability score of all clusters selected when producing a flat partition without constraints [Campello et al., 2013a]. This will give a cluster stability score for each cluster in the range [0,1]. Constraint satisfaction for each cluster can be computed as the percentage of total

constraints that are satisfied by the given cluster. This is trivial to compute for should-link constraints. Should-not link constraints are satisfied when one of the constraint objects is a member of the cluster and the other is not (this counts 0.5 towards that cluster's constraint satisfaction). Campello et al. note that, in some cases, should-not-link constraints may be satisfied by labelling objects as noise [Campello et al., 2013a, Campello et al., 2015]. For this reason, they introduce "virtual nodes". Each cluster in the cluster tree has a virtual node, and all objects which are members of the cluster but not members of the cluster's children (that is, they become noise in the hierarchy) are included in the cluster's virtual node. Should-not-link constraint satisfaction is calculated for virtual nodes, and is included with the constraint satisfaction of the cluster's descendants when determining whether to select the cluster itself or its descendants for a flat partition.

Chapter 4: An Alternative Method for Cluster Extraction (CASAR)

We now present a new approach for extracting clusters from an HDBSCAN* hierarchy that is similar to FOSC, but differs in two keys ways. Our cluster quality measure is not additive, and can select clusters in the dendrogram at any density level. We use a different local cluster quality measure than cluster stability (a density separation measure) and a different aggregation operation in the object function of the FOSC optimization problem (max or average instead of a sum). Our method, CASAR (compactness and separation adjusted ratio), is specifically designed for extracting a single density-based clusters from a subspace. It is not intended to be a general purpose replacement for cluster stability: rather, it is part of an algorithm for semi-supervised density-based projected clustering (which will be further expanded upon in Chapter 5). We first explain the theory behind our method, including its basis in Density-Based Clustering Validation (DBCV) [Moulavi et al., 2014], and then examine the strengths and weaknesses of CASAR when compared to cluster stability. We perform a number of experiments, both on synthetic and real data sets, and report the results.

4.1 CASAR

In Chapter 1, we stated that density-based clusters "are regions of high density separated by regions of low density". The cluster stability measure does not *directly* take into consideration the differences in density between clusters and the regions surrounding the clusters. Part of our proposal involves replacing the cluster stability measure with a measure that compares cluster *compactness* (that is, how similar objects within the cluster are to each other) to cluster *separation* (that is, how dissimilar objects within the cluster are to objects that are not members of the cluster). Moulavi et al. note that "the vast majority of relative validity criteria are based on the idea of computing the ratio of within-cluster scattering (compactness) to between-cluster separation" [Moulavi et al., 2014] – therefore, we elect to use these same, commonly-used measures to produce a scoring criteria. Density-Based Clustering Validaton (DBCV) is a validation index for density-based clustering that compares cluster compactness and separation [Moulavi et al., 2014]. DBCV defines compactness for a cluster c_a as the largest edge between

cluster members (objects) in the minimum spanning tree of the mutual reachability graph of c_a – in other words, this is highest minimum-eps-connectivity between any two objects in c_a. Separation is defined as the minimum mutual reachability distance between any object in c_a and any object that is a member of another cluster. Given these definitions, DBCV favours clusters that have high density and are separated from other clusters by regions of low density. It is worth noting that DBCV does not use knn density.

Given a cluster c_a at some *Eps* level in the density dendrogram, we define c_a's compactness identically to DBCV: it is the highest minimum-eps-connectivity between any two objects in c_a. We define separation slightly differently: it is the minimum mutual reachability distance between any object in c_a and any object that is not a member of c_a – in other words, it is the smallest edge in the minimum spanning tree connecting any object in c_a to any object not in c_a. This definition differs from DBCV in that we also consider non-dense (noise) objects when computing cluster separation, since it is important not only that c_a is well separated from other clusters, but from surrounding noise as well. We define a cluster quality measure for c_a as follows: *cluster quality = (separation - compactness) / separation*. Note that unlike cluster stability, which only considers clusters at their highest *Eps* level (the level at which they first exist), our measure can provide a score for a cluster at every *Eps* level at which the cluster exists. Additionally, note that our method only works when $minClSize \geq 2$, as clusters must have more than one member object in order to compute compactness, and settings $minClSize = 1$ may result in some single object clusters.

Unfortunately, our measure has one major drawback. Consider a data set with a global outlier (that is, an object that is well-separated from the rest of the data set), o_z (see Figure 4 for an example of such a data set). Let c_a be the cluster which contains every object in the data set other than o_z. Since o_z is very well separated from the rest of the data set, the difference between c_a's compactness and separation will be very large, and c_a will receive a very high cluster quality score – in fact, c_a will have the highest cluster quality score of any cluster in the data set. This is undesirable: we argue that c_a is given a high cluster quality score not because c_a is compact and well separated, but because o_z is an interesting object. If o_z were to be removed from the data set, then c_a would become the least dense level of the root cluster, and it would no longer be possible to compute a separation value for c_a. Our method's goal is to detect the best density-based clusters in a data set, not the most interesting outliers. Therefore, in cases where there is more

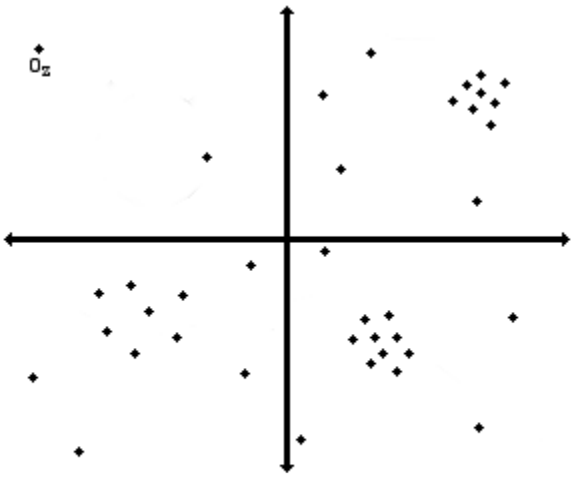

Fig 4: A 2-dimensional example data set with 3 clusters and several noise objects. The global outlier object o_z is labelled.

than one cluster in the cluster tree (that is, there is at least one cluster split), we choose to make the root cluster (at all *Eps* levels) ineligible for selection as part of the final clustering solution. This decision has the advantage of ensuring that outliers such as o_z will not influence the clusters that are selected, as well as ensuring that clusters will only be selected after a true cluster split has occurred (if one does occur in the hierarchy). In cases where there is only one cluster in the cluster tree (there is no cluster split), we apply a linear scaling factor to the cluster quality measure: *cluster quality = ((separation - compactness) / separation) * ((max edge - separation) / max edge)*. Here, *max edge* is defined as the highest minimum-eps-connectivity between any two objects in that the set. In our example, this would be the mutual reachability distance between o_z and some other object. This scaling factor biases our measure away from selecting clusters which have a high separation value. Note that, in cases where there are multiple global outliers similar to o_z, each cluster with a high separation value (that is, a separation value similar to *max edge*) will receive a low cluster quality score.

We call our modified cluster quality score CASAR (compactness and separation adjusted ratio). The CASAR score for each cluster can be computed as the dendrogram is constructed, in much the same way that cluster stability can be computed for each cluster (again, one key difference is that CASAR is calculated at every *Eps* level for each cluster). Once the dendrogram is complete, the corresponding cluster tree will also have been constructed, along with a maximum CASAR score for each cluster. At this point, provided we are attempting to extract a single cluster from a subspace, our goal will be to select the single cluster with the best CASAR

score in the cluster tree, and return this cluster.

In order for CASAR to be compatible with the FOSC framework, we also provide an aggregation method so that CASAR can be used to extract a flat partition from a dendrogram. Unlike cluster stability, CASAR is a ratio, and CASAR scores cannot simply be summed. Therefore, we propose selecting the set of clusters with the highest mean average CASAR score. At fist glance, it would appear that this set should contain only a single cluster (the cluster with the best score among all clusters). Obviously, this will result in a flat partition with the highest mean average score. However, FOSC specifies that "exactly one cluster will be selected along any branch from the root to a leaf." [Campello et al., 2013a]. This means that we cannot trivially select a partition with the highest average CASAR score by simply selecting the single cluster with the best score.

We compare the average CASAR score of descendant clusters to ancestors when determining which to select for the final flat partition. It is trivial to prove that this aggregation method will result in a final flat partition with the best overall average score. Consider a flat partition selected by our aggregation method from some dendrogram. Suppose another flat partition exists, selected from the same dendrogram, that has a higher average CASAR score. In this case, there must be some single ancestor or some set of descendants that our method could have selected to improve the overall average. However, this cannot be the case, since our method always selects descendant clusters if they have a higher average score than the ancestor cluster they are being compared to, and always selects the ancestor cluster if it has a higher score than the average score of the best set of descendants. Therefore, our method must produce a flat partition with the highest mean average CASAR score.

Our aggregation method has two important properties that FOSC requires [Campello et al., 2013a]. Firstly, it is *local*: CASAR scores for each cluster are not dependent on which clusters are selected for the flat partition. Secondly, it is compatible with the value we are trying to maximize, i.e., choosing – in a bottom-up fashion – in each subtree rooted at a node (cluster) c_a either the nodes with the highest average score selected so far in that subtree, or the node c_a, if c_a has a higher CASAR score than this average, which will result in an overall partition with the highest average score.

4.2 Conceptually Comparing CASAR and Cluster Stability

Before experimentally examining the differences between cluster stability and CASAR, it is important to discuss the conceptual differences between these two measures. Most obviously, cluster stability is designed for 'general purpose' cluster extraction in the full dimensional space, while CASAR is designed to extract a single cluster from a subspace. As has already been thoroughly discussed, CASAR *directly* takes into account the difference between a cluster's compactness and separation. Cluster stability, on the other hand, *indirectly* takes into account cluster compactness. Clusters which are more compact will 'survive' longer in the dendrogram (thereby receiving a higher cluster stability score), since they will not break apart until the *Eps* level becomes vastly lower. It is important to note that the specific results that both cluster stability and CASAR produce on a given data set are dependent on the values used for *MinPts* and *MinClSize*.

CASAR has two obvious advantages over cluster stability. First, CASAR can select each cluster at any *Eps* level, while stability only selects clusters at the level at which they first appear in the hierarchy. This means that, in most density hierarchies, CASAR has the potential to select better clusters than stability, since it is unlikely that every cluster in the cluster tree is best (or most correct, from the user's point of view) at it's highest *Eps* level. Secondly, CASAR can partition a dendrogram with just a single cluster (the root cluster) in the cluster tree, while cluster stability is unable to provide any solution for such a hierarchy (since stability is unable to select the root cluster as part of the final partition, while CASAR can select the root at some *Eps* level if there are no other clusters in the cluster tree).

Generally speaking, CASAR should outperform cluster stability on data sets that feature clusters which are both compact and well separated. Compact clusters will typically occupy a small percentage of the overall volume of the feature space. The larger the volume clusters occupy, the more difficulty CASAR will have in detecting them correctly. Note that cluster stability will also have difficulty detecting clusters with larger volume, but CASAR's performance will suffer more severely than stability's with increasing cluster volume. A given cluster's separation will be dependant on the noise objects surrounding the cluster and/or the positioning of other clusters in the data set. The average distance between clusters will typically be higher in data sets with fewer clusters, and as such, clusters will typically have better

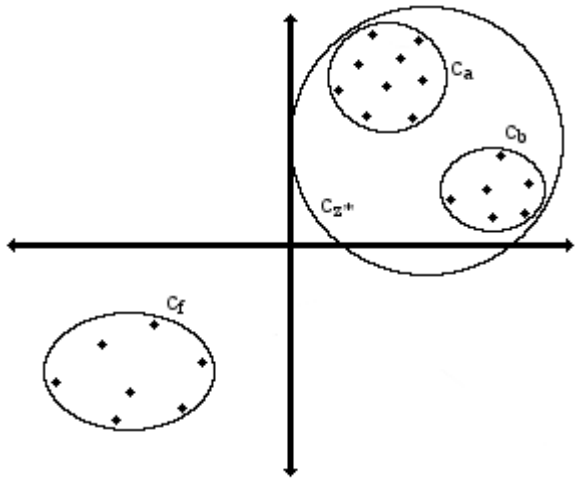

Fig 5a: A 2-dimensional data set with 3 true clusters (c_a, c_b, and c_f). The false cluster c_{z*} contains the true clusters c_a and c_b.

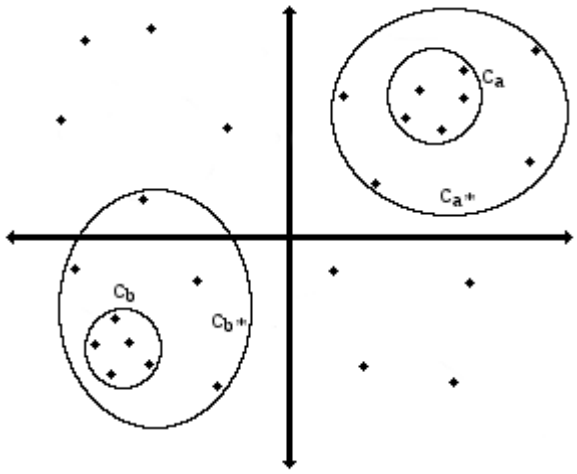

Fig 5b: A 2-dimensional noisy data set with 2 clusters. The true clusters are labelled c_a and c_b, while the noisy clusters are labelled c_{a*} and c_{b*}.

separation when there are fewer of them (and vice-versa). Figure 5a provides an example data set with clusters that are not well separated. This data set has three true clusters (c_a, c_b, and c_f) and one 'false' cluster (c_{z*}). No setting of *MinPts* and *MinClSize* will allow CASAR to correctly separate c_a and c_b, but cluster stability will correctly cluster the data set for $MinPts \in [3,6]$ and $MinClSize \in [1,6]$ or $MinPts \in [1,6]$ and $MinClSize \in [4,6]$.

Generally speaking, CASAR will perform poorly when noise objects are in close proximity to clusters – in other words, when the clusters and noise objects are not well separated. A good compactness to separation ratio may not exist for these clusters, and as such, CASAR may fail to include such clusters in a flat partition. Cluster stability will likely still be able to select the clusters (albeit marginally incorrect versions of these clusters which include noise objects). On the other hand, if the clusters are reasonably well separated from the surrounding noise, CASAR will likely be able to select noise-free versions of the clusters, while cluster stability will still select marginally incorrect, noisy versions of the clusters (since stability can only select a cluster at its highest *Eps* level in the dendrogram). Figure 5b shows an example of a noisy data set where CASAR is able to outperform cluster stability. This data set has two true clusters (c_a and c_b) and two noisy clusters (c_{a*} and c_{b*}). When $MinPts \in [3,5]$ and $MinClSize \in [1,5]$, CASAR will select the two true clusters, because they are both compact and well separated (since the noise objects surrounding each cluster have very low density). Cluster stability, on the other hand, will always choose the noisy clusters over the true clusters, as the noisy clusters both survive for a long time in the dendrogram, and are the highest *Eps* levels of their respective

clusters in the cluster tree (note that, depending on the settings, cluster stability may include additional noise objects in the clusters).

We recommend using CASAR for extracting clusters from subspaces – that is, given a relevant set of attributes for a cluster, CASAR should be able to correctly extract such a cluster from an HDBSCAN* hierarchy of the subspace. In most other cases (such as clustering in the full set of attributes), we recommend using cluster stability.

4.3 Experimentally Comparing CASAR and Cluster Stability

In this section, we compare the performance of CASAR and cluster stability across a variety of different data sets – some synthetic (artificially generated) and some real. Throughout these experiments, we vary the *MinPoints* and *MinClSize* parameters, the number and type of constraints provided, and, for synthetic data sets, we also vary features of the data set itself. These features include the number of true clusters, the number of attributes, the compactness of the clusters, and the percentage of noise objects.

4.3.1 Comparison on Synthetic Data

We construct synthetic data sets with a fixed set of features, and then vary up to one feature in each experiment. Table 1a details the different features we vary when generating synthetic data. Default values for each feature are given in bold. The number of objects, n, will adjust based on the number of clusters and the percentage of noise objects. For example, given the default values for each feature, n will be 1600. The domain of all attributes is the range [0,1], and noise objects are distributed uniformly in this range. Clusters are generated as multivariate Gaussians, with a maximum standard deviation in any attribute 1.5x times larger than the given

Data Set Feature	Values (Default Bolded)
number of clusters	1, **2**, 4, 10
d (number of attributes)	1, 2, 3, 4, 6, 8, 10, 15, 20 (**5**)
cluster minimum standard deviation	0.01, 0.02, 0.03, 0.04, **0.05**, 0.06, 0.07, 0.08, 0.09, 0.1
objects per cluster	**200**
percent noise objects	0.08, 0.25, 0.58, **0.75**

Table 1a: Features of the synthetic data sets that we generate. Default values are displayed in bold.

Algorithm Parameter	Values (Default Bolded)
MinPoints	2, 3, 4, 6, 8, **10**, 15, 20, 25
MinClSize	**10**
number of constraints	**0**, 2, 4, 6, 8, 10, 15, 20
α	**0.5**

Table 1b: Parameters settings for HDBSCAN* and FOSC when comparing CASAR and cluster stability on synthetic data. Default values are displayed in bold.

minimum standard deviation in any attribute. Note that, because these data sets are generated randomly, clusters may sometimes overlap in one or more attributes. Additionally, we ensure that cluster centers (means) are at least 2 standard deviations away from from either limit of the [0,1] range in every attribute, and we do not allow cluster members to be generated outside of the [0,1] attribute range. Each generated cluster is relevant on every attribute in the data set. The default settings are designed to approximate clusters embedded in subspaces: these settings specify a small number of compact clusters embedded in a high percentage of noise, with few attributes.

In each experiment, we vary up to one algorithm parameter. No algorithm parameters are varied when data set features are varied. Table 1b displays the parameters that we vary, along with their default values in bold. In experiments where should-link constraints are assigned, we construct the should-link constraints by randomly selecting any two objects that are members of the same cluster. Note that, by nature of this process, certain clusters may have more objects which are members of should-link constraints than other clusters. Alternatively, in experiments where should-not-link constraints are assigned, we construct the should-not-link constraints by randomly selecting any two objects that are members of different clusters. Note that we never select any noise objects for a should-not-link constraint.

Each data point in our experiments is a mean over 1000 runs. In each run, a data set is randomly generated, and then a dendrogram (cluster hierarchy) is constructed using HDBSCAN*. Both CASAR and cluster stability are used to extract a flat partition from this hierarchy. We use Adjusted Rand Index (ARI) [Hubert & Arabie, 1985] to compare the quality of the two clustering results. Rand Index [Rand, W. M., 1971] is a measure of similarity between two sets that produces a value in the range [0,1], with higher values indicating more similarity. The Rand Index is defined as $(a + b) / (a + b + c + d)$, where a is the number of pairs of objects with the same label in both the result and ground truth, b is the number of pairs of objects with different labels in both the result and ground truth, c is the number of pairs of objects with the same label in the result but different labels in ground truth, and d is the number of of pairs of objects with different labels in the result but the same label in ground truth. The Adjusted Rand Index is adjusted for chance: it produces values in the range [-1,1], with a random partition typically producing a value of approximately 0. It is important to note that, when comparing pairs of noise objects, we treat them as being members of distinct classes, rather than members of the same class.

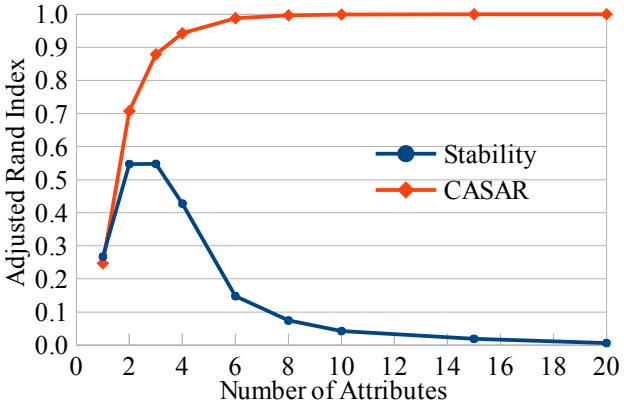

Fig 6a: Varying Number of Attributes, Number of Clusters = 1

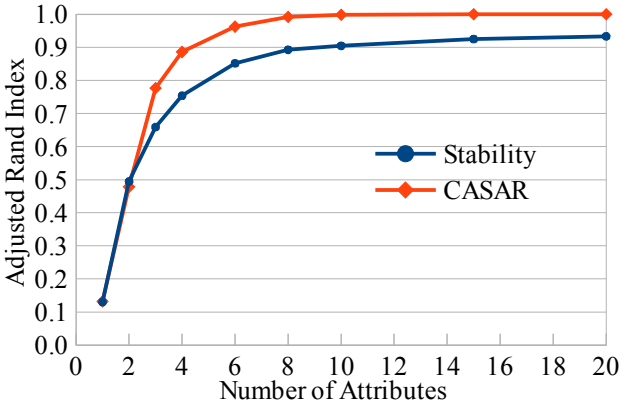

Fig 6b: Varying Number of Attributes, Number of Clusters = 2

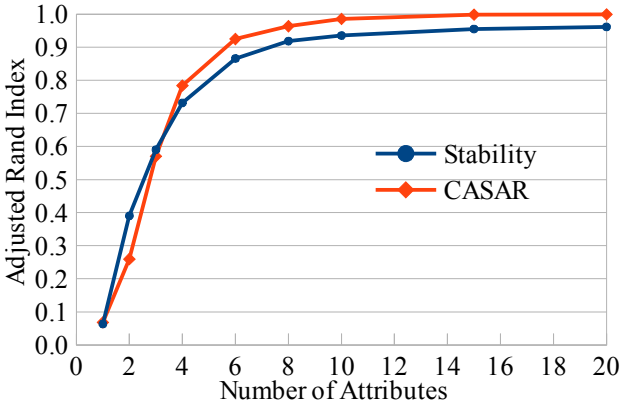

Fig 6c: Varying Number of Attributes, Number of Clusters = 4

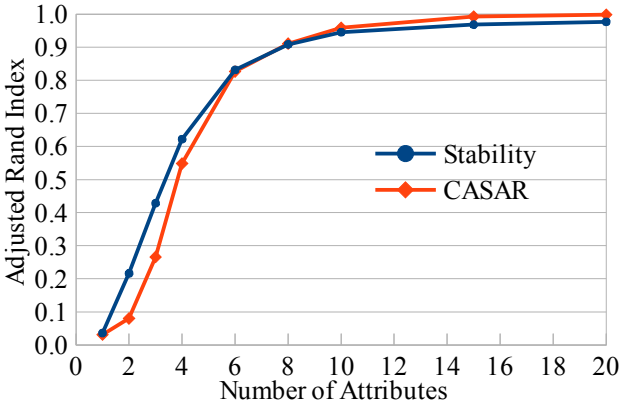

Fig 6d: Varying Number of Attributes, Number of Clusters = 10

Figures 6a through 6d display the differences in performance between cluster stability and CASAR when varying the number of clusters and the number of attributes. All other features of the data set have default values assigned (the cluster minimum standard deviation is 0.05, there are 200 objects per cluster, and there are 75% noise objects). Additionally, the algorithm parameters for HDBSCAN* used are the default values (*MinPoints* = 10, *MinClSize* = 10, there are no constraints, and $\alpha = 0.5$). For these synthetic data sets, CASAR performs better when there are fewer clusters in the data set, while cluster stability performs better when there are more clusters in the data set. In almost every case, both algorithms perform better when there are more attributes in the data set. With fewer attributes, there is less information available to differentiate cluster(s) from the surrounding noise, and cluster(s) occupy a larger percentage of the volume of the data set, meaning that some number of noise objects will be generated inside the cluster(s). As the number of attributes increases, the volume of the cluster(s) decreases, and more information becomes available to differentiate the cluster(s) from the surrounding noise.

It is worth explaining the seemingly strange ARI of cluster stability when there is 1

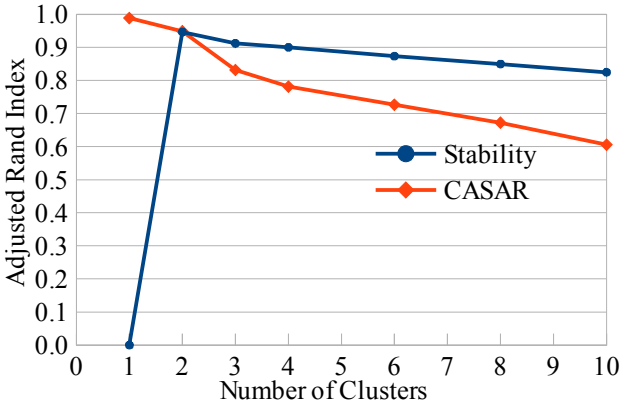

Fig 7a: Varying Number of Clusters, Percent Noise = 0.08

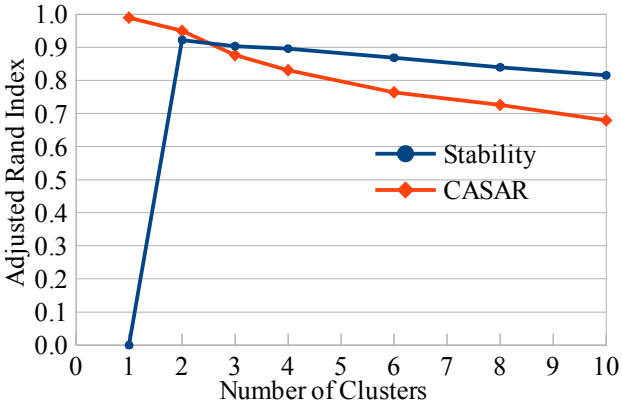

Fig 7b: Varying Number of Clusters, Percent Noise = 0.25

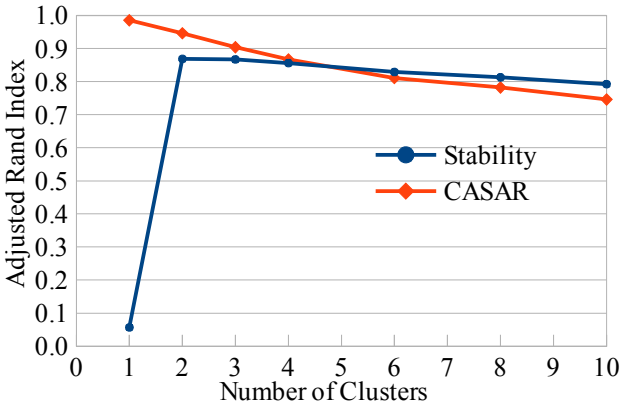

Fig 7c: Varying Number of Clusters, Percent Noise = 0.58

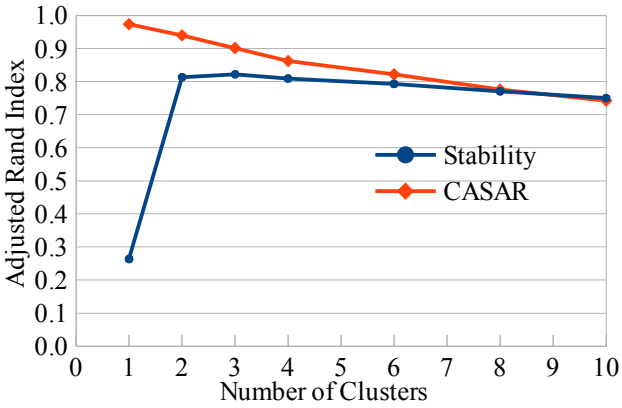

Fig 7d: Varying Number of Clusters, Percent Noise = 0.75

cluster. In higher dimensional data sets, the true cluster is well differentiated in the dendrogram, but since cluster stability cannot select the root cluster, the final partition is effectively meaningless. In the one-dimensional data set, the true cluster is difficult to detect (and likely does not exist in any meaningful form in the dendrogram), which is why both CASAR and cluster stability have a poor final partition. In the case of a data set with 2 or 3 attributes, clusters in the dendrogram other than the root exhibit some of the true cluster's structure, but are incomplete. Data sets with 2 or 3 attributes exist in a 'sweet spot' between between having a largely undetectable true cluster and having a true cluster that is so well differentiated it only appears in the root of the dendrogram.

The performance of cluster stability and CASAR when the percentage of noise and the number of clusters change is displayed in Figures 7a through 7d. All other features of the data set and all algorithm parameters are set to default values. These Figures show that CASAR is better able to handle large amounts of noise than cluster stability. Specifically, at 8% noise objects, Cluster stability outperforms CASAR as long as there are 3 or more clusters. At 75% noise

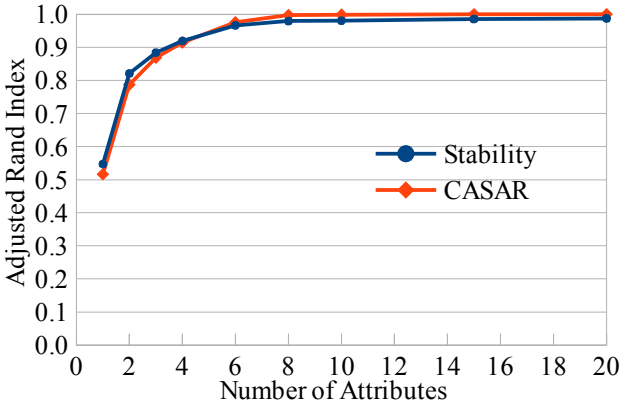

Fig 8a: Varying Number of Attributes, Percent Noise = 0.08

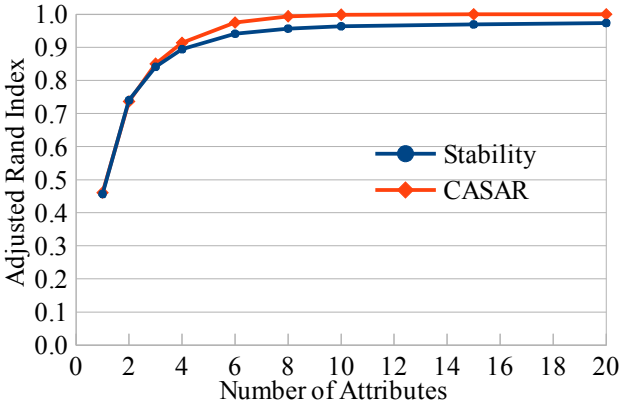

Fig 8b: Varying Number of Attributes, Percent Noise = 0.25

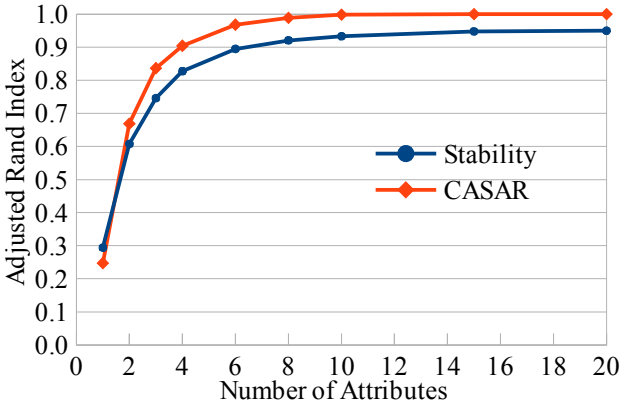

Fig 8c: Varying Number of Attributes, Percent Noise = 0.58

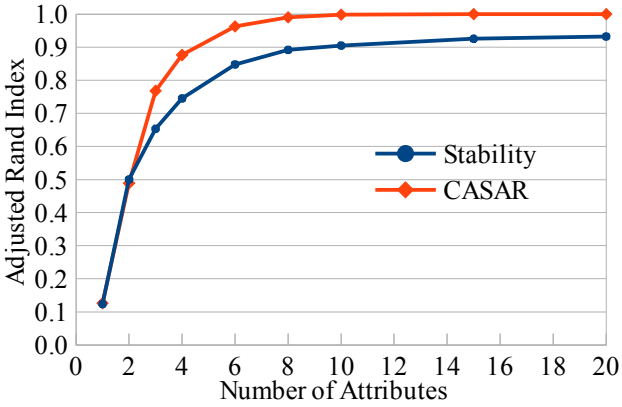

Fig 8d: Varying Number of Attributes, Percent Noise = 0.75

objects, stability only outperforms CASAR if there are 6 or more clusters. Additionally, CASAR's performance generally improves as the percentage of noise increases, while stability's performance generally degrades as the percentage of noise increases. Note that, once again, cluster stability in unable to provide a meaningful partition when there is only 1 true cluster in the data set. Stability's performance on data sets with 1 cluster does improve as the percentage of noise increases, but this is only because clusters in the dendrogram other than the root exhibit structure as the percentage of noise increases, and cluster stability is then able to select these clusters (similarly to the manner in which cluster stability behaves in Figure 6a).

Figures 8a through 8d display the performance of cluster stability and CASAR when varying the percentage of noise and the number of attributes. Once again, all other features of the data set and all algorithm parameters are set to default values. These Figures confirm that both algorithms perform better as the number of attributes in the data set increases. As Figure 7 displays, cluster stability has worsening performance relative to CASAR's performance as the percentage of noise increases. Unsurprisingly, Figure 8 shows that both algorithms are much

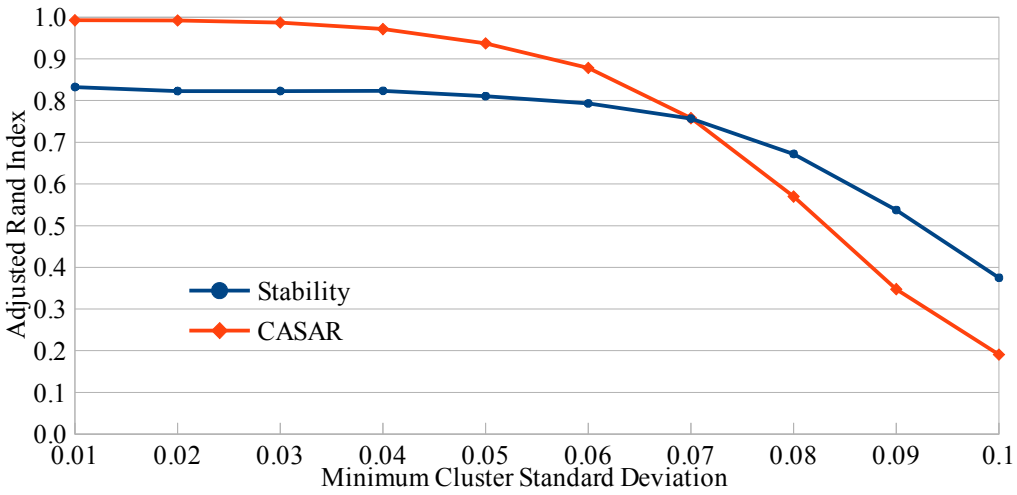

Fig 9: Varying Cluster Standard Deviation

more heavily affected by noise in lower dimensional data sets than in higher dimensional data sets. As we explained earlier in this section, with fewer attributes, clusters occupy a larger percentage of the volume of the data set, and there is less information available. Both cluster stability and CASAR can easily handle noise in higher dimensional data sets, but in lower dimensional data sets, which are already difficult to cluster correctly, increasing the percentage of noise has a more significant effect.

The differences in performance between CASAR and cluster stability as the variance (standard deviation) of the clusters changes is displayed in Figure 9. Recall that the maximum standard deviation of each cluster in any attribute is 1.5 times the minimum standard deviation in any attribute. Unsurprisingly, the performance of both algorithms degrades as the variance of clusters increases (that is, as the clusters become less dense). CASAR's performance worsens

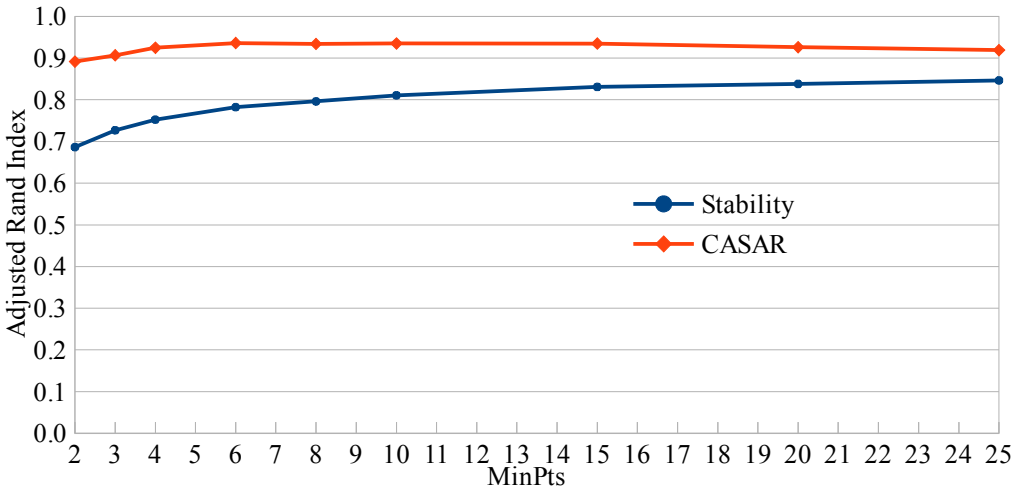

Fig 10: Varying MinPts

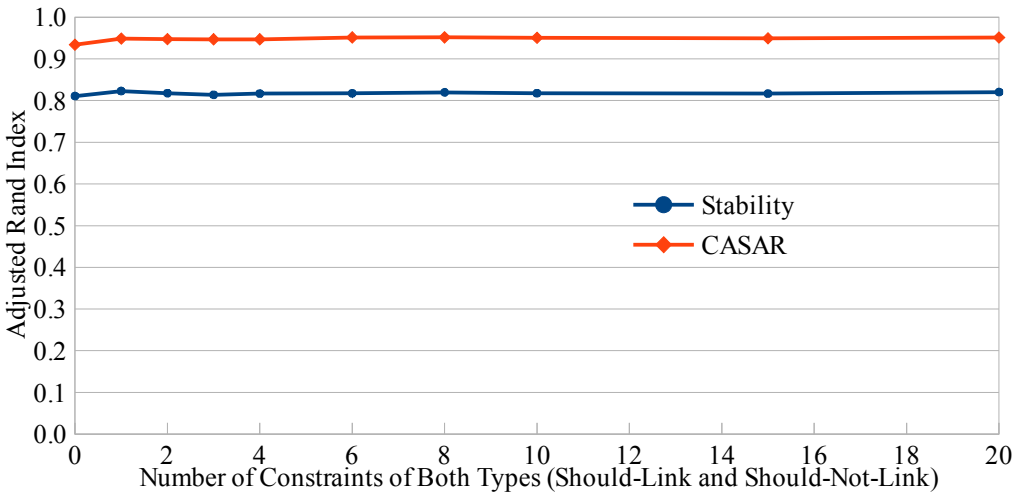

Fig 11: Varying Number of Constraints

much more significantly than stability's as the clusters become less dense. This is in line with what we stated in section 4.2 – CASAR is easily able to partition compact and well-separated clusters, but as the clusters become less and less dense, the algorithm increasingly cannot provide a meaningful partition. The same is also true for cluster stability, but with less severity.

Figure 10 shows how changing *MinPts* affects the performance of CASAR and cluster stability. Both have an 'ideal' *MinPts* setting such that performance degrades if *MinPts* is set lower or higher than this ideal setting. If *MinPts* is set too low, too few objects are considered when computing density, and density estimates will be erratic (less stable and continuous) throughout cluster(s). Setting *MinPts* too high also results in poor density estimates, since density estimates are then computed from regions which are much too large. As such, density estimates for all objects become increasingly similar. For CASAR, the ideal *MinPts* setting is somewhere in the range $MinPts \in [6,15]$, while for cluster stability, the ideal *MinPts* setting is some value greater than 25.

We perform other experiments (not pictured) where the percentage of noise in the generated data sets is varied alongside the value used for *MinPts*. Interestingly, CASAR's 'ideal' *MinPts* value is always lower than cluster stability's 'ideal' *MinPts* value (as is the case in Figure 10). Roughly speaking, CASAR searches for cluster borders where the local density changes rapidly, while cluster stability searches for large regions of approximately continuous density. As such, CASAR performs better when local density is measured – which requires a small value for *MinPts* – while cluster stability performs better when a larger region is considered when computing density – which requires a larger value for *MinPts*.

In Figure 11, we see the effect of using both should-link and should-not-link constraints on CASAR and on cluster stability. In this Figure, values along the x-axis represent the number of both constraints (that is, a value of 4 means that 4 should-link and 4 should-not-link constraints were provided when running both algorithms). Both CASAR and cluster stability show little change in performance on the synthetic data sets we generate as the number of constraints increases. This may be a result of selecting the constraints randomly (which in some cases will result in constraints that are not beneficial to either algorithm), or it may be a result of the type of data sets that we generate. Note that we also perform other experiments (not pictured) using only should-link constraints and using only should-not-link constraints, but the results are extremely similar.

The experiments we have presented here confirm what we stated in section 4.2: CASAR is not a general purpose replacement for cluster stability, but rather is an alternative to FOSC for extracting clusters from an HDBSCAN* hierarchy when attempting to find clusters in subspaces. Unlike cluster stability, CASAR can successfully partition data sets with only a single true cluster. CASAR performs best on data sets with a small number of tightly distributed clusters surrounded by a large number of noise objects. In most cases, we expect to find very few density-based clusters in any given subspace. We expect these clusters to contain a relatively small percentage of objects in the data set – meaning that the remaining objects in the subspace will be noise. Additionally, density-based clusters which are embedded in subspaces will often be tightly distributed (relative to the volume of the subspace).

4.3.2 Comparison on Real Data

In this section, we compare CASAR and cluster stability on 4 well-known data sets from the UCI machine learning repository [Lichman, 2013]. These data sets are ‘ecoli’, ‘glass identification’, ‘iris’, and ‘wine.’ There are 336, 215, 150, and 178 objects in these data sets, respectively. Additionally, these data sets have 8, 6, 3, and 3 clusters, respectively, and 7, 8, 4, and 13 attributes, respectively. None of these data sets have any noise objects (that is, all objects belong to a cluster). We chose to set *MinClSize* = 10 for both CASAR and cluster stability, and vary *MinPts*. Note that both the ecoli data set and the glass identification data set contain clusters with less than 10 objects, however, we still find *MinClSize* = 10 to generally produce the best results for both algorithms across all data sets.

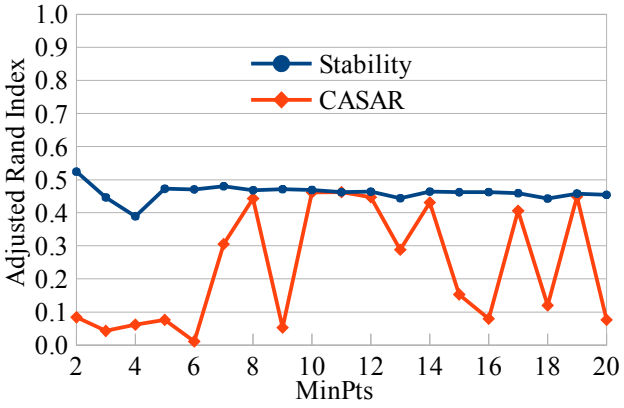

Fig 12a: Ecoli Data Set

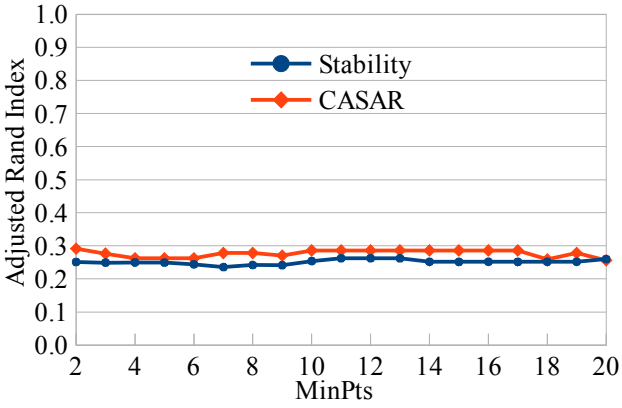

Fig 12b: Glass Identification Data Set

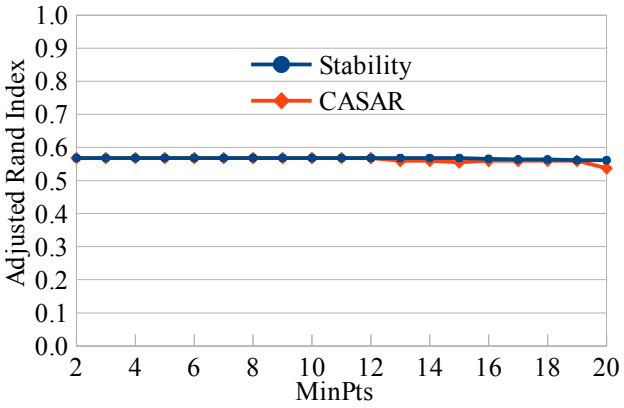

Fig 12c: Iris Data Set

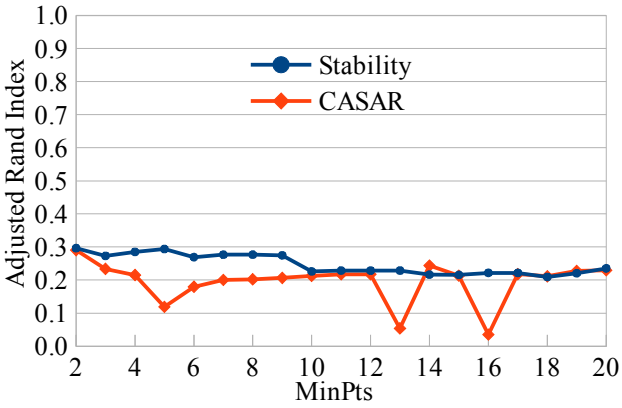

Fig 12d: Wine Data Set

Figure 12a displays the differences in performance between CASAR and cluster stability on the ecoli data set as *MinPts* changes. Obviously, stability's performance on this data set is much more stable. Additionally, cluster stability outperforms CASAR on almost every setting of *MinPts*. CASAR sometimes incorrectly selects descendant clusters in the cluster tree instead of their more correct ancestors. At other times, CASAR selects clusters at density levels which result in very few objects being labelled (that is, CASAR finds the best compactness to separation ratio for these clusters at density levels such that most of the objects in these clusters become noise). In both cases, CASAR clusters far fewer objects than cluster stability does. Generally speaking, the dendrogram (density hierarchy) produced by HDBSCAN* has too few clusters in it (compared to the true number of clusters) for the ecoli data set. Specifically, for $MinPts \in [7,20]$, the dendrogram only contains 3 clusters (one of which is the root).

Results for the glass identification data set are given in Figure 12b. On the glass identification data set, both algorithms have fairly consistent performance, and CASAR generally slightly outperforms cluster stability. For $MinPts \in [2,9]$, CASAR selects a slightly different set

of clusters than stability (preferring smaller descendant clusters over ancestors). For *MinPts* ∈ [10,20], there are only 3 clusters in the dendrogram, and CASAR and cluster stability select the same clusters, but CASAR selects these clusters at a different density level. As was the case with the ecoli data set, the dendrogram produced by HDBSCAN* typically has too few clusters in it when compared to the true number of clusters.

Figure 12c displays results for the iris data set. CASAR and cluster stability have practically identical performance on this data set. Both algorithms combine two of the true clusters into a single cluster (largely as a result of the dendrogram produced by HDBSCAN*).

Results for the wine data set are displayed in Figure 12d. Cluster stability outperforms CASAR on this data set, and also performs more consistently. As was the case with the ecoli data set, CASAR sometimes selects incorrect descendant clusters, and sometimes selects the same clusters as stability, but at an incorrect density level.

Overall, these results indicate that CASAR prefers smaller clusters than cluster stability. CASAR sometimes selects descendants when cluster stability would select ancestors, and at other times, CASAR selects the same clusters as stability, but at a higher density level (which results in smaller versions of the same clusters). In the case of the ecoli and wine data sets, this makes CASAR's performance unstable and inferior to cluster stability's performance, while in the case of the glass identification data set, CASAR outperforms cluster stability slightly. Across all data sets, both CASAR and cluster stability rely heavily on the quality of the dendrogram produced by HDBSCAN*. In many cases, it is not possible for either algorithm to produce a high quality result, given the possible partitions that can be selected from the dendrogram.

We also perform a number of experiments using should-link and should-not-link constraints on the ecoli, glass identification, iris, and wine datasets. For cluster stability, adding constraints does not significantly alter performance on any of the 4 data sets. CASAR shows little change in performance on 3 of the datasets, but adding constraints does significantly improve CASAR's performance on the ecoli data set. Figure 13 displays this experiment. Note that *MinClSize* = 10. In cases where CASAR performs very poorly, adding a small number of should-link and should-not-link constraints causes significant improvement. Note that cluster stability's performance without constraints is still superior to CASAR's performance with constraints on the ecoli data set.

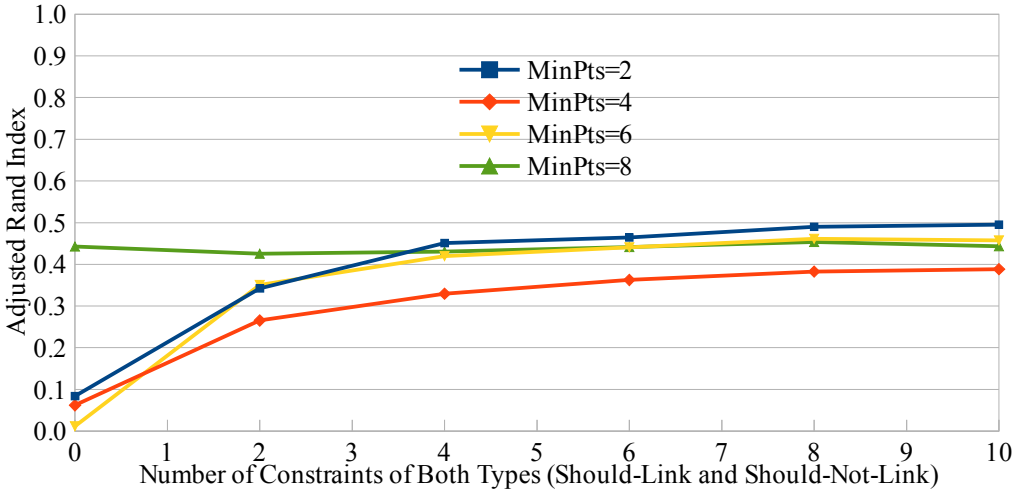

Fig 13: Varying Number of Constraints for CASAR, Ecoli Data Set

4.3.3 Summary of Results

At first glance, the experiments we have presented in this Chapter might seem to suggest that CASAR is inferior to cluster stability. It is important to re-iterate that CASAR is specifically designed for finding density-based clusters in subspaces. Our experiments on real data sets reinforce that CASAR is not designed as a general purpose replacement for cluster stability. Additionally, these experiments show that both algorithms are very dependent on the dendrogram produced by HDBSCAN*. These experiments also demonstrate that, in cases where CASAR has very poor performance, adding a small number of constraints can significantly improve CASAR's performance. Our experiments on synthetic data sets show that CASAR performs well on data sets that have a small number of very compact clusters embedded in a large percentage of noise. In general, density-based clusters embedded in subspaces are likely to be small, compact, and surrounded by large amounts of noise. As such, CASAR should excel at extracting density-based clusters from subspaces.

Chapter 5: Density-Based Attribute Selection Using Constraints

In chapters 1 and 2 we discussed the 'curse of dimensionality'. We now present a method for finding a relevant subspace for a density-based cluster in a data set, given a set of should-link objects has been provided (pairwise should-link constraints exist between every pair of objects in this set). We assume that this set of should-link constraints consists only of pairs of objects that belong to the same cluster – this cluster will be constructed once the subspace is found. Our method is quite general, and can be used as a pre-processing step for any number of density-based clustering algorithms. For the given set of should-link constraints, we formally define the best subspace (most relevant) as the subspace which contains a density-based cluster that satisfies all of the pairwise constraints and gives the highest possible CASAR score. This definition should be useful for most (if not all) density-based clustering algorithms, as it prioritizes finding subspaces that contain cluster(s) which are regions of high density separated by regions of low density. After presenting our method for semi-supervised attribute selection, we explain how combining this method with CASAR results in a semi-supervised density-based projected clustering algorithm.

5.1 Theoretical Basis for Attribute Selection

One obvious way to find such a subspace would be to compute an HDBSCAN* hierarchy and then run our modified version of FOSC (CASAR) on every possible combination of dimensions. We could then examine all clusters which satisfy all of the pairwise should-link constraints, and select the combination of attributes (subspace) which contains the cluster with the best CASAR score among all these clusters. It should be apparent that such a solution is computationally infeasible, as it requires CASAR to be run 2^d - 1 times. Therefore, for the sake of computationally efficiency, our method is an approximate method. We select a subspace by examining properties of the set of constraints (since the cluster is not yet known, and these constraints are meant to be representative of the cluster).

A distinction should be made between relevant attributes (attributes which, if included in the relevant subspace, will result in a cluster with a higher CASAR score being found) and

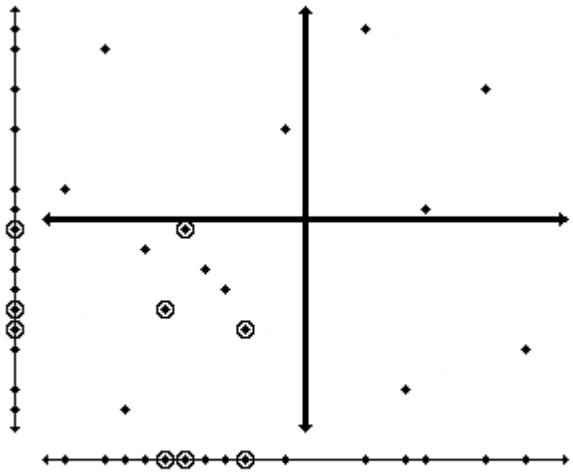

Fig 14a: A 2-dimensional relevant subspace with 1 cluster. Should-link objects are highlighted, with objects projected onto 1-dimensional projections.

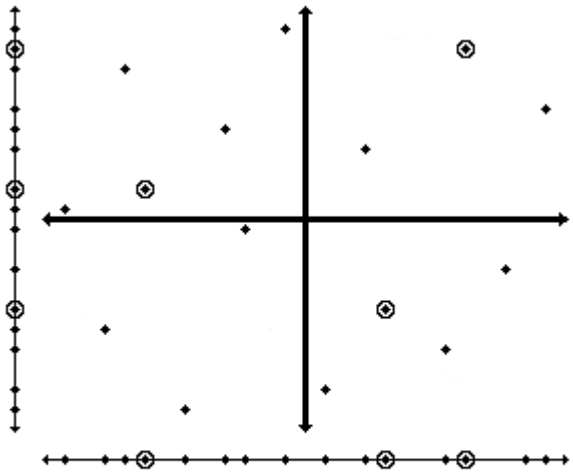

Fig 14b: A 2-dimensional irrelevant subspace. Should-link objects are highlighted, with objects projected onto 1-dimensional projections.

irrelevant attributes (attributes which will worsen the CASAR score if included in the relevant subspace). Generally speaking, relevant attributes help 'clarify' the cluster (that is, make the cluster structure easier to detect for density-based clustering methods, by way of improving the cluster's compactness and separation ratio), while irrelevant attributes do the opposite. It is non-trivial to determine whether attributes are relevant or irrelevant for a given set of should-link constraints – in fact, we usually cannot make a binary relevant / irrelevant distinction (largely because we do not fully know the cluster yet). Our goal, therefore, is firstly to compute a relevancy score for all attributes for the given set of should-link constraints, and secondly to declare some set of attributes relevant (thereby forming the relevant subspace) based on the relevancy scores of these attributes.

We assume that pairs of objects in the set of should-link constraints have significantly better density-connectivity (lower minimum-eps-connectivity) on relevant attributes than on irrelevant attributes. Consider a subspace that consists of relevant attributes only, such as the subspace in Figure 14a. Such a subspace contains some cluster which satisfies all of the should-link constraints. Since this cluster is a dense region, the set of should-link objects have low minimum-eps-connectivity to each other (signifying strong density-connectivity), since the objects are both individually dense and connected to each other by a region of high density. As Figure 14a demonstrates, these objects will also have low minimum-eps-connectivity to each other on the individual attributes that make up the subspace. Alternatively, consider a subspace that consists of several irrelevant attributes only, such as the subspace in Figure 14b. The set of

should-link objects will have poor density-connectivity (high minimum-eps-connectivity) in this subspace, since they are not part of the same cluster. Most (of it not all) of the objects are not individually dense, and few (if any) of the objects are connected by regions of high density. As Figure 14b demonstrates, the set of should-link objects will also have high minimum-eps-connectivity on the individual attributes that make up this subspace. Therefore, we can construct a relevant subspace by finding attributes where the set of should-link objects have low minimum-eps-connectivity.

A few clarifying statements need to be made about our theoretical basis. Firstly, in order to compare minimum-eps-connectivity across different attributes, the range of values in each attribute needs to be normalized. When the range of values in each attribute is not normalized, distances cannot be directly compared – meaning that density measures (such as core distance, mutual reachability distance, and minimum-eps-connectivity) cannot be directly compared either. Any value range will be sufficient, provided all attributes are normalized to the same range. Secondly, it is important to acknowledge that it is possible for objects in a pairwise should-link constraint to have low minimum-eps-connectivity on an irrelevant attribute – any number of examples of this behaviour can be constructed. However, the larger the set of should-link objects, the less likely it is that all pairs of objects in the set have low minimum-eps-connectivity on the same irrelevant attribute. This means that, while some irrelevant attributes may appear relevant for a few pairs of should-link objects, they are unlikely to appear relevant for all (or even most) of the pairs of should-link objects. Thirdly, it is possible for pairs of should-link objects to have poor density-connectivity (high minimum-eps-connectivity) on relevant attributes. Figure 15 provides an example of such an occurrence: the 2 should-link objects are members of the same cluster in the subspace but have high minimum-eps-connectivity on both of the attributes that make up the subspace. This situation is more likely to occur in data sets where clusters are not aligned along the attributes (such as in Figure 15). Of course, the minimum-eps-connectivity of pairs of objects in any cluster will be most heavily influenced by the density (and particularily the compactness) of that cluster.

It is worth noting that we do not make use of should-not link constraints, as it is difficult to ascertain how to use them in a data set with multiple clusters in different subspaces. Does specifying a should-not link constraint indicate that certain objects should never be clustered together in any subspace, or does it indicate that certain objects should not be clustered together

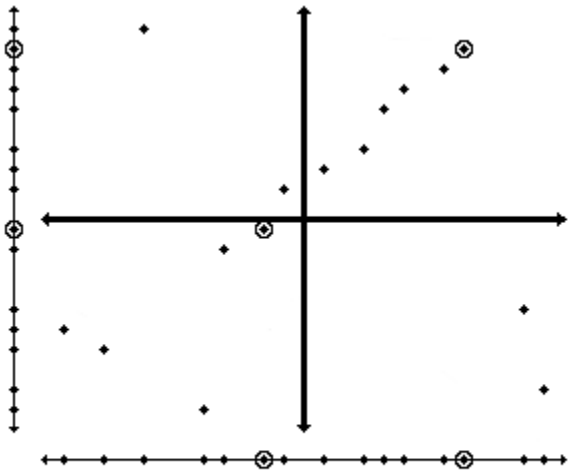

Fig 15: A problematic 2-dimensional subspace with 1 cluster. The highlighted should-link objects have poor density-connectivity in 1D projections.

in a specific subspace? In the former case, should-not link constraints become extremely restrictive. It is highly unlikely for a user to be able to state with confidence that a certain set of objects can never be clustered together in any subspace. In the latter case, the constraint can either be fulfilled trivially, by finding any one subspace where the objects are not clustered together (in which case the constraint is essentially useless), or else the user must specify which subspace(s) or attribute(s) the should-not link constraint is relevant for. It is highly unlikely that, prior to performing any clustering, the user is able to provide this type of information.

5.2 Method Overview

Our method is complex, and involves several steps. Before explaining the specific details of each step, we will provide an overview of our method as a whole:

Step 1: Construct a one-dimensional minumum spanning tree (more specifically, a minimum spanning tree of the mutual reachability graph) on each attribute of the data set. These one-dimensional minimum spanning trees allow the minimum-eps-connectivity between pairs of objects on each individual attribute to be computed quickly.

Step 2: Select an initial subspace. A relevancy score is computed for each attribute, and then a statistical test (comparing these relevancy scores to a null distribution) is used to determine which attributes to select as the initial subspace. These relevancy scores are computed using the minimum-eps-connectivity between pairs of should-link objects.

Step 3: Expand the set of should-link objects. Additional objects are added to the set of

should-link objects by finding objects in the subspace which have low mutual reachability distance to should-link objects.
Step 4a: Remove any irrelevant attributes from the initial subspace. We determine which attributes are irrelevant by performing a statistical test that uses covariance matrix determinants.
Step 4b: Add any relevant attributes that were initially rejected. Once again, we perform a statistical test using covariance matrix determinants to determine which attributes should be added to the final subspace.

5.3 Computing Minimum-Eps-Connectivity in Individual Attributes

In this section, we will show that the minimum-eps-connectivity for a pair of objects in a single attribute can be computed in $O(n)$ time – provided a one-dimensional minimum spanning tree has been constructed on that attribute. More specifically, this one-dimensional minimum spanning tree is a minimum spanning tree of a mutual reachability graph on a single attribute. We will also show that it takes $O(n \log n)$ time to construct such a one-dimensional minimum spanning tree.

For a given attribute a, the first step of this construction process is to sort the objects on a (o_{1a}, o_{2a}, o_{3a}, … o_{na}) in ascending order. A wide variety of well-known algorithms exist to perform this sorting in $O(n \log n)$ time. Next, the core distance of each object on attribute a is calculated in $O(n\ MinPts)$ time using a 'sliding window'. Consider an object o_x on attribute a (o_{xa}) – $CoreDist_{MinPts}(o_{xa})$ is determined by o_{xa}'s $MinPts$-nearest-neighbors. This set of objects must be contiguous on the sorted list of objects and must include o_{xa}. If a non-contiguous set of objects were selected, some object not included in the set would be closer to o_{xa} than some object already in the set. Additionally, o_{xa} must be included in the set, as $Dist(o_{xa},o_{xa}) = 0$. Therefore, one of the contiguous sets of $MinPts$ objects which includes o_{xa} on the sorted list of objects must be o_{xa}'s $MinPts$-nearest-neighbors. For every object on attribute a, every such candidate $MinPts$-nearest-neighbors set can be examined using a sliding window of width $MinPts$. This window always contains $MinPts$ objects, starting with the $MinPts$ objects with the lowest values in the sorted list. Let o_{la} be the lowest value object in the window, and o_{ha} be the highest value object in the window. For each object o_{xa} in the window, we set o_{xa}'s 'candidate core distance' to be $maximum(Dist(o_{xa},o_{la}), Dist(o_{xa},o_{ha}))$. This candidate core distance is then compared to o_{xa}'s

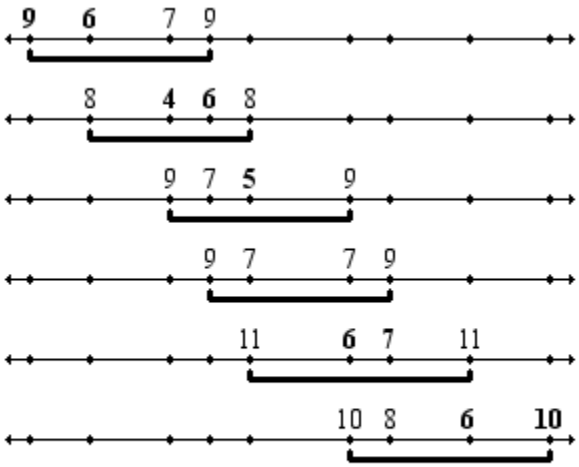

Fig 16: An example sliding window iterating over an attribute. Candidate core distances (*MinPts* = 4) are displayed, with final core distances bolded.

current core distance – if the candidate core distance is lower, then it replaces the current core distance. After a candidate core distance has been computed for each of the *MinPts* objects in the window, the window slides – o_{la} is removed, and the next object in the sorted list (the lowest value object that has not yet been in the window) is added to the window. Additionally, the labels o_{la} and o_{ha} are re-assigned. Candidate core distances are then computed for this new set of objects. This process is iteratively repeated until a core distance has been computed for all n objects in attribute a (which takes $O(n\ MinPts)$ time). See Figure 16 for an example of this process.

Building a one-dimensional minimum spanning tree (a minimum spanning tree of a mutual reachability graph for a given attribute) is non-trivial for most values of *MinPts*. Recall that Prim's algorithm [Prim, R. C., 1957] can be used to construct a minimum spanning tree. Prim's algorithm begins to construct the minimum spanning tree by selecting any arbitrary vertex and adding it to the tree. Next, the edge with minimal weight among all edges which connect a vertex already in the tree and a vertex not yet in the tree is attached to the minimum spanning tree (thereby attaching the corresponding vertex to the minimum spanning tree as well). This process is iteratively repeated until all vertices are attached to the minimum spanning tree. Recall that the mutual reachability distance of two objects o_y and o_z is defined as $maximum(CoreDist_{MinPts}(o_y), CoreDist_{MinPts}(o_z), Dist(o_y, o_z))$. Additionally, recall that when $MinPts \leq 2$, the mutual reachability distance of any two objects will simply be the distance between the objects. Therefore, when $MinPts \leq 2$, the one-dimensional minimum spanning tree can be constructed by simply

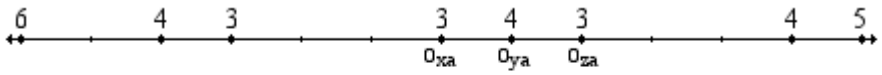

Fig 17: A single attribute '*a*' with 8 objects. Core distances are displayed for *MinPts* = 4. Consecutive 'ticks' are 1 unit distance apart.

connecting objects which are immediate neighbors (since immediate neighbors will have minimal edge length to each other in the minimum spanning tree). When $MinPts \geq 3$, we cannot simply connect objects and their immediate neighbors, and the process becomes more complicated.

Consider the example attribute *a* provided in Figure 17, with 8 objects, and core distances calculated for all objects with *MinPts* = 4. Suppose we begin to construct the minimum spanning tree of the mutual reachability graph with object o_{xa}. One of the objects with lowest mutual reachability distance to o_{xa} is o_{za}, since $MRD_{MinPts}(o_{xa},o_{za}) = 3$. When we connect o_{za} to the minimum spanning tree by adding an edge between o_{xa} and o_{za}, we 'skip over' o_{ya}. Note that, in this example, no object left of o_{xa} or right of o_{za} needs to share an edge with o_{ya} in the minimum spanning tree. Additionally, o_{ya} has minimal mutual reachability distance to both o_{xa} and o_{za}, and therefore an edge can correctly be added between o_{ya} and o_{xa} or between o_{ya} and o_{za}. This example holds true in the more general case: given some attribute *a* and objects o_x, o_y, and o_z such that $o_{xa} < o_{ya} < o_{za}$, $CoreDist_{MinPts}(o_{xa}) < CoreDist_{MinPts}(o_{ya})$, $CoreDist_{MinPts}(o_{za}) < CoreDist_{MinPts}(o_{ya})$, and given o_{xa} and o_{za} share an edge in the minimum spanning tree of the mutual reachability graph (they have minimal mutual reachability distance to each other), o_{ya} will only share an edge with either o_{xa} or o_{za} in the minimum spanning tree. It is important to note that $CoreDist_{MinPts}(o_{ya}) > CoreDist_{MinPts}(o_{xa})$ and $CoreDist_{MinPts}(o_{ya}) > CoreDist_{MinPts}(o_{za})$ must be true, and that $Dist(o_{xa},o_{za}) < CoreDist_{MinPts}(o_{ya})$ must also be true – otherwise, there could not be an edge between o_{xa} and o_{za}, because o_{ya} would have a lower mutual reachability distance to at least one of them.

We first prove that no objects have minimal mutual reachability distance to o_{ya} only. For each object o_{la} left of o_{xa} (objects with lower value than o_{xa}), $MRD_{MinPts}(o_{la},o_{xa}) \leq MRD_{MinPts}(o_{la},o_{ya})$. This holds true because $Dist(o_{la},o_{xa}) < Dist(o_{la},o_{ya})$ and $CoreDist_{MinPts}(o_{xa}) < CoreDist_{MinPts}(o_{ya})$. Parallel arguments can be made to show that for each o_{ra} right of o_{za} (objects with value higher than o_{za}) $MRD_{MinPts}(o_{ra},o_{za}) \leq MRD_{MinPts}(o_{ra},o_{ya})$. For each object o_{ma} between o_{xa} and o_{za}, $MRD_{MinPts}(o_{ma},o_{xa}) \leq MRD_{MinPts}(o_{ma},o_{ya})$ and $MRD_{MinPts}(o_{ma},o_{za}) \leq MRD_{MinPts}(o_{ma},o_{ya})$, since $CoreDist_{MinPts}(o_{xa}) < CoreDist_{MinPts}(o_{ya})$ and $CoreDist_{MinPts}(o_{za}) < CoreDist_{MinPts}(o_{ya})$ and $Dist(o_{xa},o_{za}) < CoreDist_{MinPts}(o_{ya})$ – which means that $Dist(o_{ma},o_{xa}) < CoreDist_{MinPts}(o_{ya})$ and $Dist(o_{ma},o_{za}) <$

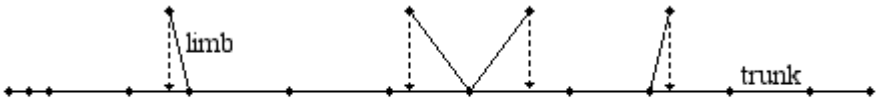

Fig 18: An example minimum spanning tree of a mutual reachability graph on an individual attribute. Objects in 'limbs' are raised for clarity.

$CoreDist_{MinPts}(o_{ya})$.

Secondly, we prove that that o_{ya} has mutual reachability to o_{xa} and o_{za} which is less than or equal to o_{ya}'s mutual reachability distance to any other object. By definition, o_{ya}'s mutual reachability distance to any other object must be at least equal to $CoreDist_{MinPts}(o_{ya})$. We know that $CoreDist_{MinPts}(o_{ya}) \geq CoreDist_{MinPts}(o_{xa})$ and that $CoreDist_{MinPts}(o_{ya}) \geq CoreDist_{MinPts}(o_{za})$. Additionally, $Dist(o_{xa},o_{za}) < CoreDist_{MinPts}(o_{ya})$. Therefore, o_{xa} and o_{za} are both within o_{ya}'s core distance of o_{ya}, and have minimal mutual reachability distance to o_{ya}.

For a given attribute a, we construct a one-dimensional minimum spanning tree starting with the object o_{xa} with the lowest core distance in attribute a. By examining up to *MinPts* - 1 objects right of o_{xa} (objects with higher value than o_{xa}) we find all of the objects with minimal mutual reachability distance to o_{xa} among all objects right of o_{xa}. From among these objects we select the object o_{za} with minimal core distance, and we then connect o_{za} and o_{xa}. Any objects 'skipped over' by this edge are connected to o_{xa} (as proven valid in the previous paragraph). The same process is then repeated from o_{za}, and iteratively repeated until all objects right of o_{xa} are connected to the minimum spanning tree. All objects left of o_{xa} can be connected to the minimum spanning tree in the same manner. Constructing a one-dimensional minimum spanning tree using this procedure takes $O(n\ MinPts)$ time, and results in a minimum spanning tree that has a 'trunk' spanning the entirety of the attribute along with 'limbs' of single objects that connect to the trunk. Figure 18 gives shows an example of the structure of such a minimum spanning tree.

Computing the minimum-eps-connectivity between any two objects in a one-dimensional minimum spanning tree is an $O(n)$ operation. This operation involves traversing through the minimum spanning tree from one object to the other object, keeping track of the maximum edge encountered along the path between these objects. When computing the minimum-eps-connectivity for several should-link constraints, we can reduce the running time of this operation by storing the minimum-eps-connectivity of pairs of objects on the trunk of the minimum spanning tree. These stored minimum-eps-connectivity values essentially act as 'shortcuts' – when the path between two objects in a constraint includes the path between a pair of 'shortcut

objects' on the trunk, the section of the trunk between these shortcut objects will not need to be traversed over. Instead, the stored minimum-eps-connectivity value can be used to represent that section of the trunk. As we compute the minimum-eps-connectivity for each pairwise should-link constraint, we store the maximum edge encountered along the trunk between the leftmost and rightmost objects on the trunk for that constraint. Subsequent constraints are then able to make use of this shortcut.

5.4 Selecting a Subspace

At this point, our goal is to use the minimum-eps-connectivity values computed on each attribute for every pair of should-link objects to select some number of attributes as the initial subspace. We want to combine the minimum-eps-connectivity values computed on each attribute into a single 'relevancy score', so that we can compare the relevancy scores between attributes, and determine which attributes are relevant and which are irrelevant. A relevancy score for each attribute is computed by summing the minimum-eps-connectivity of every pair of should-link objects on the attribute, and then taking the reciprocal (multiplicative inverse) of this sum. We choose to sum pairwise minimum-eps-connectivity values on each attribute so that each of these values is given equal 'weight' in the attribute's relevancy score. Furthermore, we take the reciprocal of this sum total so that more relevant attributes have higher relevancy scores, while less relevant attributes have lower relevancy scores. Next, the attributes are sorted in order from best (highest relevancy score) to worst (lowest relevancy score). At this point, some number of the best scoring attributes should be selected as the subspace – however, it is non-trivial to decide how many attributes to select.

A threshold scoring value could be set by the user, and all attributes with a higher relevancy score would become part of the subspace – however, such a threshold would be extremely difficult to set, as relevancy scores do not have any inherent meaning, and will likely vary greatly from one data set to another (and vary with the number of should-link objects). Another option would be to have the user set the number of attributes to select – unfortunately, this value would also be difficult to set, as different sets of should-link constraints (representing different clusters) will likely have different numbers of meaningful attributes. Additionally, it is unlikely that the user has any knowledge of how many attributes should be relevant for a given set of should-link objects. Finally, the user could examine the sorted attributes in order to

manually select the subspace – this option is prohibitively expensive for more than a small number of sets of should-link constraints, and also requires the user to have some understanding of what the attribute relevancy scores mean.

Given these issues, we choose to use a statistical test to determine which attributes should be selected as the subspace. For each given attribute, our null hypothesis states that the attribute is irrelevant. We compare each attribute's relevancy score to a null distribution of relevancy scores, and then declare an attribute to be statistically significant (as well as relevant) if it has a score in the top *RandMECThresh* percent of the distribution. In this statistical test, *RandMECThresh* is the significance level. The null distribution consists of relevancy scores of randomly generated, uniformly distributed attributes where should-link objects are randomly selected. Note that each of these randomly generated attributes consists of n objects. Additionally, the sets of should-link objects which are randomly selected contain the same number of objects as the set of should-link objects provided by the user.

This statistical test requires a large number of random attributes to be generated, and a one-dimensional minimum spanning tree to be constructed on each one. Constructing a one-dimensional minimum spanning tree on an attribute takes $O(n \log n)$ time. Note that we can improve the efficiency of the statistical test by caching these one-dimensional minimum spanning trees and then re-using them with different sets of user-defined should-link objects. In order to do this, *MinPts* and n must remain the same. In cases where the number of should-link objects in the set also stays the same, the null distribution can be cached and re-used instead. It is important to note that, in cases where less than 2 attributes are declared relevant by the statistical test, we choose to declare the 2 attributes with best relevancy score as relevant. Constructing a subspace with 0 or 1 attributes is not reasonable.

At first glance, it would appear that the null distribution could instead be produced by randomly generating a single uniform noise attribute and then computing relevancy scores for a large number of random should-link object sets on this single attribute. Unfortunately, this method does not produce an equivalent null distribution to generating multiple random attributes and computing a relevancy score for a single should-link set on each attribute. Both methods produce null distributions with the same mean average relevancy score, but as the number of should-link objects in the set increases, the variance in the null distribution generated from multiple attributes increases more rapidly than the variance in the null distribution generated from

a single attribute. On any given attribute, multiple pairs of should-link objects will have the same minimum-eps-connectivity. The paths that connect these pairs of objects in the minimum spanning tree of the mutual reachability graph all pass through the same edge with maximal edge weight. Several such locally maximal edges will exist. Therefore, computing a null distribution of relevancy scores from a single attribute will result in the same minimum-eps-connectivity values being combined over and over again (resulting in lower variability), while computing a null distribution from multiple attributes will result in a wider variety of minimum-eps-connectivity values being combined (resulting in higher variability).

Our statistical test requires a user input parameter, *RandMECThresh*. This parameter is very easy to set and understand, as it specifies a statistical significance level. It is worth noting that *RandMECThresh* also specifies the level of type 1 error (false positives). The user must also specify the number of random attribute relevancy scores to generate. A large number of random attribute relevancy scores should be generated, such that the distribution of relevancy scores generated is stable and statistically meaningful. In our experiments, we generate 1000 relevancy scores for null distributions.

The number of irrelevant attributes incorrectly selected by the statistical test will depend on the number of irrelevant attributes in the data set and the statistical significance level *RandMECThresh*. In addition to these type 1 errors (false positives), we expect some small number of type 2 errors (false negatives) to occur – that is, some number of relevant attributes may be rejected by the statistical test. Given these issues, we wish to 'refine' the selected subspace, by attempting to identify and remove irrelevant attributes (if any have been selected) and add relevant attributes (if any have not yet been selected). As previously discussed in section 6.1, our ability to correctly select a subspace improves as the number of should-link objects increases. Therefore, in order to improve the subspace, we use the currently selected subspace (which we assume is 'mostly correct') to select more should-link objects, and then use this expanded set of should-link objects to refine the subspace. In cases where the initial subspace is not ‘mostly correct’, expanding the set of should-link objects poses little risk. For such a subspace, performing clustering will likely result in a low quality result. As such, there is little risk in selecting additional should-link objects, given that the end clustering result, with or without refining the subspace, will likely be quite poor.

5.5 Expanding the Set of Should-Link Objects

As previously stated in Chapter 1, density-based clusters are “regions of high density separated by regions of low density”. In the relevant subspace, cluster members have strong density-connectivity to each other, and lower density-connectivity to other objects. Therefore, given a 'mostly correct' subspace, we expect cluster members to have relatively strong density-connectivity to each other in this subspace. Hence, objects with the strongest density-connectivity (lowest minimum-eps-connectivity) to should-link objects are very likely to also be cluster members. Based on this principle, we select additional should-link objects by determining which objects have the lowest minimum-eps-connectivity to the set of should-link objects in the subspace. For each should-link object in the current set, we find a set of *NumSLExpand* objects that have low minimum-eps-connectivity to that should-link object. Note that we do not include any objects already in the should-link set in this set of *NumSLExpand* objects. We then combine all of these sets of *NumSLExpand* objects and select the 'best' *NumSLExpand* objects from this combined set. The 'best' objects selected from this combined set have low combined minimum-eps-connectivity to the set of should-link objects as a whole. This final set of *NumSLExpand* objects is then added to the current set of should-link objects.

We could compute the minimum-eps-connectivity of every object in the data set to every object in the set of should-link objects, but such an action can be computationally expensive, as it requires constructing a minimum spanning tree of the mutual reachability graph in the current subspace, an $O(n^2)$ operation. Therefore, we choose to use an approximate method. For each should-link object, we find the *NumSLExpand* objects that are not already in the should-link set and have the lowest mutual reachability distance to the should-link object. We use mutual reachability distance instead of minimum-eps-connectivity for two reasons. First of all, mutual reachability distance is faster to compute. Secondly, in cases where two objects are within either object's core distance of each other, the mutual reachability distance and the minimum-eps-connectivity between the objects will be the same. We prove this statement in the following paragraphs.

Setting the *NumSLExpand* parameter too high increases the probability that objects which are not cluster members may be selected for the expanded set of should-link objects. As such, we recommend setting the *NumSLExpand* parameter equal to *MinPts*. This ensures that most of the *NumSLExpand* objects selected for each should-link object are members of the should-link

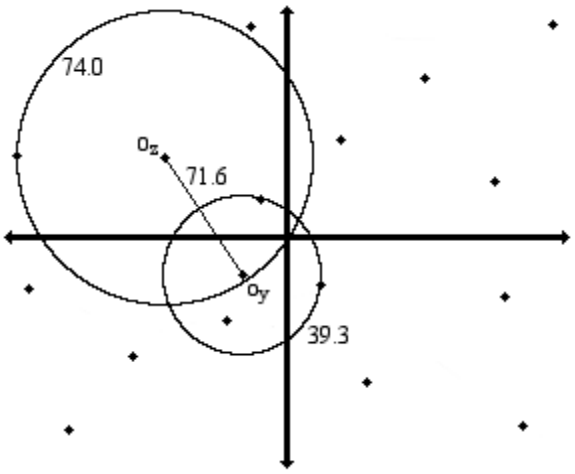

Fig 19: A 2-dimensional example data set, with core distances for objects o_z and o_y when *MinPts* = 4 and distance between o_z and o_y displayed.

object's *MinPts*-nearest-neighbors.

Consider Figure 19. Given two objects o_y and o_z such $Dist(o_y,o_z) < CoreDist_{MinPts}(o_z)$, and $CoreDist_{MinPts}(o_z) > CoreDist_{MinPts}(o_y)$, we can conclude that $MRD_{MinPts}(o_y,o_z) = CoreDist_{MinPts}(o_z)$. This is trivial to prove, as mutual reachability distance is defined as $maximum(CoreDist_{MinPts}(o_y), CoreDist_{MinPts}(o_z), Dist(o_y, o_z))$. Furthermore, we can prove that $MEC_{MinPts}(o_y,o_z) = MRD_{MinPts}(o_y,o_z)$. $MEC_{MinPts}(o_y,o_z)$ is defined as the maximal edge in the path connecting o_y and o_z in the minimum spanning tree of the mutual reachability graph. Given that these edges are mutual reachability distances, any edge directly connected to o_z must minimally have a weight at least equal to $CoreDist_{MinPts}(o_z)$. Since the edge between o_y and o_z in the mutual reachability graph has the lowest possible weight of any edge connected to o_z, this edge can be selected for the minimum spanning tree. Therefore, the path between o_y and o_z in the minimum spanning tree consists of a single edge, and the weight of this edge is $MRD_{MinPts}(o_y,o_z)$, which is consequently equal to $MEC_{MinPts}(o_y,o_z)$.

In cases where $CoreDist_{MinPts}(o_y) > Dist(o_y,o_z)$, $MEC_{MinPts}(o_y,o_z) = MRD_{MinPts}(o_y,o_z)$ (symmetric arguments to the ones already given prove this statement). Therefore, for a given should-link object o_y, this means that all objects within o_y's Eps-neighborhood (the *MinPts* - 1 objects closest to o_y, other than o_y itself) have minimum-eps-connectivity to o_y equal to their mutual reachability distance to o_y. Additionally, there will likely also be some number of objects just outside o_y's Eps-neighborhood for which the mutual reachability distance to o_y and the minimum-eps-connectivity to o_y are the same (such as o_z in Figure 19). This is why, for a given

Object	o_y	o_{19}	o_{52}	o_{64}	o_{21}	o_{60}	o_{13}	o_{39}	o_{33}	o_{55}	o_{77}	o_{29}
Distance to o_y	0.0	2.2	2.8	3.6	4.4	5.4	5.8	6.7	7.6	8.1	8.5	8.5

etc...

Fig 20: A partial example sorted list L_y of object / distance pairs for a should-link object o_y in a subspace.

should-link object o_y, we choose to aproximate o_y's minimum-eps-connectivity to nearby objects by simply computing the mutual reachability distance to these objects. For many of these objects, mutual reachability distance to o_y and minimum-eps-connectivity to o_y will be the same.

For each given should-link object o_y in the set of should-link objects, we begin the process of finding the *NumSLExpand* objects with lowest mutual reachability distance to o_y in the current subspace by computing $CoreDist_{MinPts}(o_y)$. To compute o_y's core distance, we calculate o_y's distance to every object o_1, o_2, ... o_n (including o_y itself) in the current subspace, and then store distance / object pairs in a list, $LDist_y$. We then sort $LDist_y$ in ascending order of distance. The distance of the *MinPts* entry in the list will be equal to $CoreDist_{MinPts}(o_y)$. This entire process takes *O(n log n)* time, as computing the distances between o_y and all objects in the subspace is an *O(n)* time operation, while sorting $LDist_y$ is an *O(n log n)* time operation. See Figure 20 for a partial example of a sorted list. For the example $LDist_y$ given in Figure 20, $CoreDist_3(o_y) = 2.8$, and $CoreDist_5(o_y) = 4.5$. Note that we cache $CoreDist_{MinPts}(o_y)$ once we have computed this value.

In order to find the *NumSLExpand* objects with the lowest mutual reachability distance to o_y in the subspace, we iteratively examine entries in $LDist_y$, computing the mutual reachability distance between o_y and the objects in these entries. We iterate through $LDist_y$ in ascending order of distance, and ignore any objects that are already members of the set of should-link objects (including o_y itself). As we iterate, we maintain a set of objects with the lowest mutual reachability distance to o_y found so far – $SMRD_y$. This set is updated as necessary when examining new entries. Note that there may be more than *NumSLExpand* objects in $SMRD_y$ at times, as multiple objects may have the same mutual reachability distance to o_y. We choose to not arbitrarily select some objects for $SMRD_y$ and reject others when all of these objects have the same mutual reachability distance to o_y.

At first glance, it would appear necessary to iterate through all *n* entries in $LDist_y$ in order to find the *NumSLExpand* objects with lowest mutual reachability distance to o_y. Fortunately, this is not the case. Suppose we are iteratively examining objects in $LDist_y$, and there are already *NumSLExpand* (or more) objects in $SMRD_y$. Let o_z be the object in $SMRD_y$ with highest mutual reachability distance to o_y among all objects currently in $SMRD_y$. Given an object o_x in $LDist_y$

such that $Dist(o_x,o_y) > MRD_{MinPts}(o_z,o_y)$, we know that $MRD_{MinPts}(o_x,o_y) > MRD_{MinPts}(o_z,o_y)$, since $MRD_{MinPts}(o_x,o_y) \geq Dist(o_x,o_y)$ be definition. Therefore, o_x will not be added to $SMRD_y$, and furthermore, we do not need to compute $MRD_{MinPts}(o_x,o_y)$. Moreover, since $LDist_y$ is sorted by distance from o_y, no object after o_x in $LDist_y$ will have lower mutual reachability distance to o_y than the objects already in $SMRD_y$. Therefore, we do not need to iterate through all n objects in $LDist_y$ to find the *NumSLExpand* objects with best mutual reachability distance to o_y. Once an object o_x is found in $LDist_y$ such that $Dist(o_x,o_y) > MRD_{MinPts}(o_z,o_y)$ (provided $SMRD_y$ already has at least *NumSLExpand* objects in it, and o_z is the object with highest mutual reachability distance to o_y among objects in $SMRD_y$), we can stop iterating over $LDist_y$.

In order to compute the mutual reachability distance between o_y and some other object o_z in the subspace, we must compute $Dist(o_y,o_z)$, $CoreDist_{MinPts}(o_y)$, and $CoreDist_{MinPts}(o_z)$. Two of these values, $Dist(o_y,o_z)$ and $CoreDist_{MinPts}(o_y)$, will already be known when examining o_z – therefore, we only need to compute $CoreDist_{MinPts}(o_z)$. Given that we only need to find the *NumSLExpand* objects with lowest mutual reachability distance to o_y, any such o_z is likely to be 'relatively nearby' o_y in the subspace. This means that we can make use of the sorted list $LDist_y$ in order to improve the efficiency of computing $CoreDist_{MinPts}(o_z)$. More specifically, we can use the distances in $LDist_y$ (distances between o_y and other objects) along with the triangle inequality to provide a lower bound on o_z's distance to other objects, thus limiting the number of objects we need to examine when computing $CoreDist_{MinPts}(o_z)$.

In order to compute $CoreDist_{MinPts}(o_z)$, we maintain a set of the *MinPts* closest objects to o_z among objects that have been examined so far – $SDist_z$. Our goal is to find o_z's *MinPts*-nearest-neighbors, since $CoreDist_{MinPts}(o_z)$ is defined as o_z's distance to its *MinPts*-nearest-neighbor. We begin this process by examining all of the entries in $LDist_y$ up to and including o_z's entry in $LDist_y$, computing the distance between o_z and these objects (and updating $SDist_z$). If we have not yet examined *MinPts* entries (equivalently, if $SDist_z$ does not yet contain at least *MinPts* objects), we continue to examine entries in $LDist_y$ until we have examined *MinPts* entries. At this point, $SDist_z$ will contain at least *MinPts* objects, including o_z itself. Furthermore, for each remaining object o_x in $LDist_y$ yet to be examined, $Dist(o_x,o_y) \geq Dist(o_z,o_y)$.

Consider Figure 21a, where $Dist(o_y,o_z) = 171$ and $Dist(o_y,o_x) = 197$ have been computed, but $Dist(o_z,o_x) = 106$ has not yet been computed. Given the triangle inequality, we know that $Dist(o_y,o_x) \leq Dist(o_z,o_x) + Dist(o_y,o_z)$. If we substract the $Dist(o_y,o_z)$ factor from both sides, we get

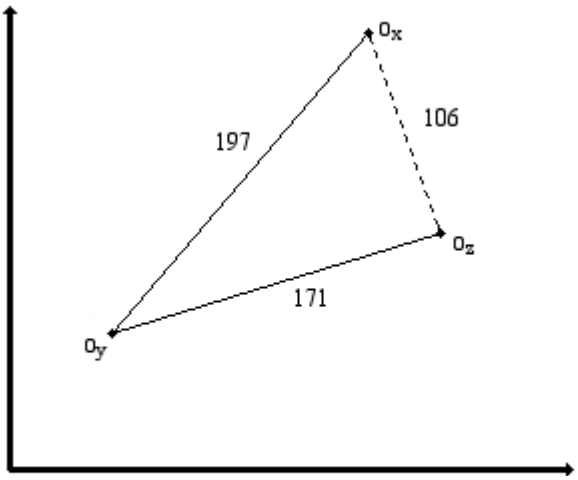

Fig 21a: Three objects (o_x, o_y, and o_z) in a 2-dimensional space, with distances between the objects displayed.

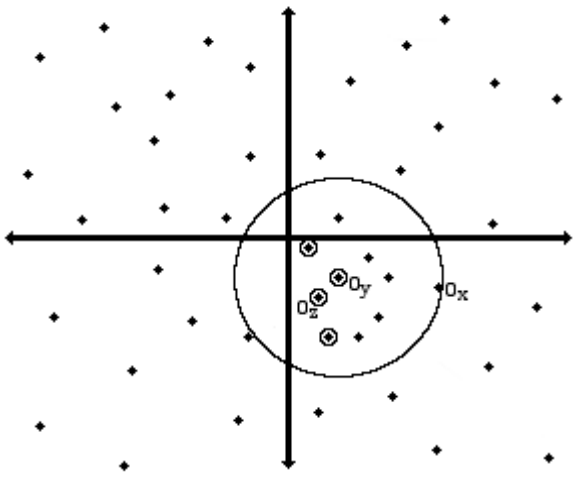

Fig 21b: An example of pruning when finding o_z's 4-nearest-neighbors (which are highlighted). Only objects in the circle are examined (including o_x).

$Dist(o_y,o_x) - Dist(o_y,o_z) \leq Dist(o_z,o_x)$. Therefore, $Dist(o_y,o_x) - Dist(o_y,o_z)$ gives us a lower bound on the unknown distance $Dist(o_z,o_x)$. For the example given in Figure 21a, we can determine that $Dist(o_z,o_x) \geq 197\text{-}171 = 26$. Using this lower bound, we can limit the number of entries we need to examine in $LDist_y$ when computing $CoreDist_{MinPts}(o_z)$. Specifically, once $SDist_z$ contains at least *MinPts* objects including o_z, we can compute $Dist(o_y,o_x) - Dist(o_y,o_z)$ for each object o_x we encounter while iterating through $LDist_y$. If, for any such o_x, the lower bound on $Dist(o_z,o_x)$ (that is, $Dist(o_y,o_x) - Dist(o_y,o_z)$) is greater than o_z's highest distance to any object in $SDist_z$, we can stop iterating – $SDist_y$ contains o_z's *MinPts*-nearest-neighbors. This is easy to prove: o_x cannot be one of o_z's *MinPts*-nearest-neighbors (given the lower bound on $Dist(o_z,o_x)$), and since $LDist_y$ is sorted by distance from o_y, all of the remaining objects in $LDist_y$ cannot be part of o_z's *MinPts*-nearest-neighbors, either. Figure 21b provides an example of this pruning process. At this point, we can easily compute $CoreDist_{MinPts}(o_z)$, and we can also trivially compute $MRD_{MinPts}(o_z,o_y)$. It is important to note that we cache $CoreDist_{MinPts}(o_z)$ after computing this value. We use this cached value rather than recomputing $CoreDist_{MinPts}(o_z)$ when needed. Specifically, we may need $CoreDist_{MinPts}(o_z)$ when finding the objects with lowest mutual reachability distance to other should-link objects and / or when combining the sets of objects with lowest mutual reachability distance into a final set.

Using the methods we have just described, we can compute a set of *NumSLExpand* (or more) objects with lowest mutual reachability distance to each should-link object in the set of should-link objects. Our next step is to combine each of these sets into a final set of

NumSLExpand objects with lowest combined mutual reachability distance to the set of should-link objects. For each object o_z that is a member of one or more sets with lowest mutual reachability distance, we want to compute o_z's 'combined mutual reachability distance'. This is done by summing o_z's mutual reachability distance to each should-link object. We choose to sum these values so that each should-link object has equal 'weight' in affecting o_z's combined mutual reachability distance. Oftentimes, for a particular should-link object o_y, $MRD_{MinPts}(o_y,o_z)$ will not have been computed, because o_z is not a member of o_y's set of objects with lowest mutual reachability distance. In these cases, we need $Dist(o_y,o_z)$, $CoreDist_{MinPts}(o_y)$, and $CoreDist_{MinPts}(o_z)$ in order to compute $MRD_{MinPts}(o_y,o_z)$. Given that $CoreDist_{MinPts}(o_z)$ and $CoreDist_{MinPts}(o_y)$ are cached, looking these values up is trivial. These means that we only need to compute $Dist(o_y,o_z)$, which is a constant time operation.

Once we have computed the combined mutual reachability distance for each such object o_z, we select the *NumSLExpand* objects with the lowest combined mutual reachability distance, and add these objects to the existing set of should-link objects. Note that we may select more than *NumSLExpand* objects to add to the set of should-link objects, as multiple objects may have the same combined mutual reachability distance (and we do not want to select some objects and arbitrarily reject others when these objects have the same combined mutual reachability distance). Given that we have expanded the set of should-link objects, we will now use this expanded set to refine the subspace.

5.6 Refining the Subspace

We have stated multiple times that density-based clusters are “regions of high density separated by regions of low density”. In the relevant subspace, we expect the cluster we are searching for to be dense (compact). Refer to Figure 22a for an example of such a cluster. This cluster is 'tightly' distributed – that is, there is low variance in the cluster's distribution about it's center. In an irrelevant subspace, such as the example given in Figure 22b, the cluster has no structure, and cluster members are distributed 'widely' throughout the subspace. As such, the cluster distribution exhibits much higher variance. Generally speaking, a density-based cluster should have much lower variance in its relevant subspace than in any other subspace. By extension, the expanded set of should-link objects should also have lower variance in the relevant subspace than in any other subspace.

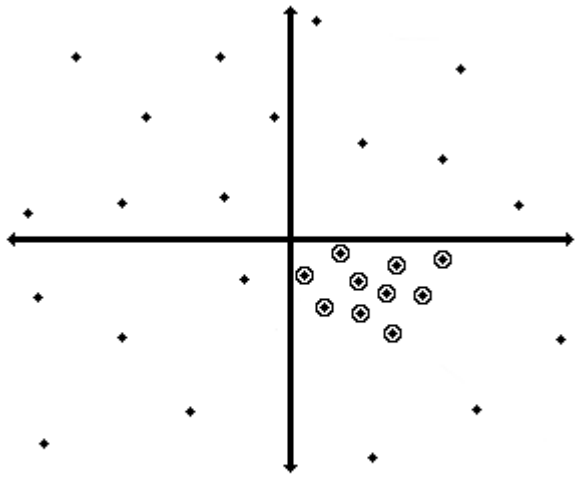

Fig 22a: A density-based cluster in a 2-dimensional relevant subspace, with cluster members highlighted.

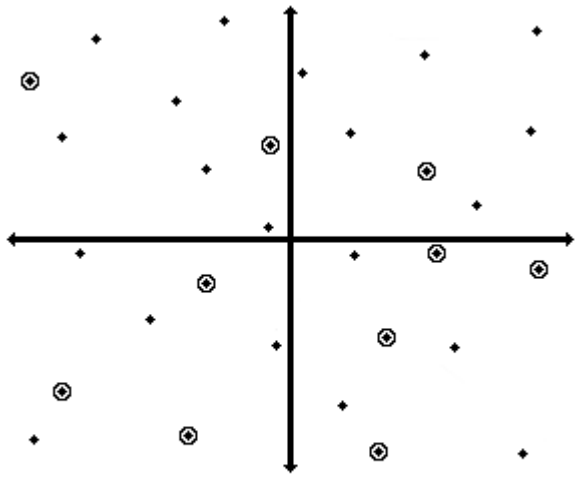

Fig 22b: A density-based cluster in a 2-dimensional irrelevant subspace, with cluster members highlighted.

It is trivial to represent the distribution of a cluster with a covariance matrix, which is a symmetric positive semi-definite matrix. Entries in such a covariance matrix represent the covariance of the cluster between different attributes. The entry in the *j*th row and *k*th column of the matrix is the covariance of the cluster between the *j*th and *k*th attributes of the subspace. An identical entry is found in the *k*th row and *j*th column of the matrix, since it is symmetric. Diagonal entries in the matrix represent the variance of the cluster on individual attributes. Such a covariance matrix has several important properties. The eigenvalues of this matrix represent the variance of the cluster in orthogonal directions. These directions are given by the corresponding eigenvectors, and are usually not axis-parallel. As with any square matrix, the determinant of this matrix is equal to the product of the eigenvalues. In other words, for a covariance matrix of a cluster, the determinant is equal to the product of the variances of the cluster distribution in orthogonal directions. Therefore, we can use the determinant of the covariance matrix of a cluster as a measure of the quality of the subspace that the cluster is in. By extension, the determinant of the covariance matrix of the set of should-link objects can be used as a measure of subspace quality.

We choose to use the determinant of the covariance matrix as a measure of subspace quality for several reasons. Mostly obviously, the determinant should be smaller in the relevant subspace than in irrelevant subspaces. Computing the determinant of a matrix is a well known problem, and many algorithms for computing the determinant exist. Finally, the matrix determinant is related proportionally to the volume of the cluster – smaller clusters will have

Object	Attr. 4	Attr. 11	Attr. 15
o_{12}	0.1	0.2	0.1
o_{37}	-0.1	0.3	0.3
o_{45}	-0.5	0.6	0.4
o_{48}	0.2	0.2	0.2
o_{98}	-0.3	0.4	0.5

Covariance Matrix

	Attr. 4	Attr. 11	Attr. 15
Attr. 4	0.066	-0.037	-0.030
Attr. 11	-0.037	0.022	0.016
Attr. 15	-0.030	0.016	0.020

Fig 23: An example with 5 should-link objects in 3 attributes, and the covariance matrix that represents the distribution of objects.

lower determinants, and larger clusters will have higher determinants. Computing the determinant is superior to computing the volume of a bounding box enclosing the cluster, as individual outliers can drastically affect the volume of the bounding box, but will have less impact on the cluster's covariance matrix determinant. Therefore, we use the determinant of the covariance matrix of the expanded set of should-link objects to refine the subspace. We first perform a statistical test to remove any irrelevant attributes from the initial subspace, and then perform a second statistical test to add any relevant attributes that were missed when constructing the initial subspace. An example covariance matrix, generated from 5 objects in a subspace of 3 attributes, is given in Figure 23. It is worth noting that the size of the covariance matrices we use will be dependent on the number of attributes in the initial subspace. In some cases, if subspaces become very large, our statistical tests may become computationally infeasible.

We will first explain our statistical test for removing irrelevant attributes using covariance matrix determinants. For each given attribute in the initial subspace, our null hypothesis states that the attribute is irrelevant. We perform this test by generating a unique null distribution of covariance matrix determinants for each attribute in the initial subspace, and then comparing the covariance matrix determinant of the should-link set to this null distribution. The unique null distribution for each attribute is generated by temporarily re-assigning all of the should-link objects to random locations in the attribute being examined, updating the covariance matrix, and then computing the determinant of the updated covariance matrix. This process is repeated a given number of times to generate the null distribution. In other words, the null distribution consists of determinants that were computed after replacing the attribute being examined with uniformly distributed noise. If the should-link set's determinant is in the bottom *RandDetThresh* of the null distribution, the attribute being examined is considered statistically significant (as well as relevant). Otherwise, the attribute is declared irrelevant (and will be removed from the

Object	Attr. 4	Attr. 11	Rand
o_{12}	0.1	0.2	0.6
o_{37}	-0.1	0.3	1.0
o_{45}	-0.5	0.6	-0.8
o_{48}	0.2	0.2	0.3
o_{98}	-0.3	0.4	-0.4

Covariance Matrix

	Attr. 4	Attr. 11	Rand
Attr. 4	0.066	-0.037	0.125
Attr. 11	-0.037	0.022	-0.080
Rand	0.125	-0.080	0.430

Fig 24: The example covariance matrix given in Figure 23, except that attribute 15 has been replaced with random values.

subspace once all attributes in the subspace have been examined). Figure 24 gives an example of replacing attribute 15 with random uniform noise values. In order to construct a null distribution for attribute 15, this replacement would be repeated a number of times, and the determinants of the resulting covariance matrices would be saved. The determinant of the covariance matrix in Figure 23 would then be compared to this null distribution of determinants.

After all of the irrelevant attributes have been removed, we perform a second statistical test to determine if any of the attributes that were not included in the initial subspace should be deemed relevant (and added to the subspace). For each given attribute that was not included in the initial subspace, our null hypothesis states that the attribute is irrelevant. We begin this test by generating a single null distribution of covariance matrix determinants. This distribution is generated by temporarily adding a noise attribute to the subspace (that is, an attribute in which the should-link objects are randomly uniformly distributed), updating the covariance matrix, and then computing the determinant of the updated covariance matrix. The temporary noise attribute is then removed. Repeating this process a given number of times generates the null distribution. For each attribute that is being examined, the attribute is temporarily added to the subspace, the covariance matrix is updated, the determinant is computed, and the attribute is removed from the subspace. Each attribute's respective determinant is then compared to the null distribution. If any attribute has a determinant in the bottom *RandDetThresh* of the null distribution, the attribute is considered statistically significant, and declared relevant. All of the relevant attributes are added to the subspace after every attribute that was not included in the initial subspace has been examined. Figure 25 gives an example of adding a random uniform noise attribute to the subspace. In order to construct a null distribution, this replacement would be repeated a number of times, and the determinants of the resulting covariance matrices would be saved. Each attribute being examined would be added to the subspace in the same manner, and for each such

Object	Attr. 4	Attr. 11	Attr. 15	Rand
o_{12}	0.1	0.2	0.1	-0.5
o_{37}	-0.1	0.3	0.3	0.6
o_{45}	-0.5	0.6	0.4	-0.2
o_{48}	0.2	0.2	0.2	0.0
o_{98}	-0.3	0.4	0.5	0.4

Covariance Matrix

	Attr. 4	Attr. 11	Attr. 15	Rand
Attr. 4	0.066	-0.037	-0.030	-0.019
Attr. 11	-0.037	0.022	0.016	0.004
Attr. 15	-0.030	0.016	0.020	0.032
Rand	-0.019	0.004	0.032	0.158

Fig 25: The example covariance matrix given in Figure 23, except that a random attribute has been added.

attribute, the determinant of the resulting covariance matrix would be compared to the null distribution of determinants just generated.

It is important to note that, in some cases, it may not be possible to compute the determinant of a covariance matrix (depending on the number of should-link objects that were used to make the covariance matrix). In these cases, we cannot refine the subspace. Furthermore, if removing irrelevant attributes would result in a subspace with less than 2 attributes, we do not remove any of the attributes that have been deemed irrelevant.

It is worth noting that in both of these statistical tests, we do not completely re-construct the covariance matrix when modifying the subspace (either by temporarily adding an attribute or temporarily re-assigning the locations of should-link objects in an attribute). Rather, we adjust only the single row and column for which the covariance and variance have changed (as show in Figures 26 and 27). Both of these statistical tests are relatively time efficient. The matrices being operated on are relatively small (with the number of rows and columns equal to the number of attributes in the subspace). Additionally, the set of should-link objects should be relatively small, meaning that computing variances and covariances will be very fast, and generating the null determinant distributions should be fast.

5.7 A Semi-Supervised Density-Based Projected Clustering Method

We will now explain how our method for semi-supervised attribute selection can be combined with CASAR to produce a semi-supervised density-based projected clustering algorithm. Our clustering algorithm requires set(s) of should-link objects provided by the user. Each one of these set(s) corresponds to a cluster that has yet to be discovered. For each such set, we find the relevant subspace using our method for attribute selection. Then, we generate

pairwise should-link constraints between every pair of objects in the set. We also generate pairwise should-not-link constraints between every object in the set of should-link objects currently being examined and every object in the other set(s). These constraints are generated in order to improve CASAR's ability to extract the correct cluster from the subspace. Next, we construct an HDBSCAN* hierarchy in the relevant subspace, and then run CASAR on this hierarchy (making use of the should-link and should-not-link constraints we have constructed). Rather than extracting a flat partition from the hierarchy, CASAR selects the single cluster with the best score in the cluster tree, and associates this cluster with the subspace. This process is then repeated for all of the remaining set(s) of should-link objects. Our method's final output is a set of potentially overlapping clusters, each of which is associated with a potentially overlapping subspace.

Our semi-supervised density-based projected clustering method has 6 input parameters: *RandMECThresh*, *NumSLExpand*, *RandDetThresh*, *MinPts*, *MinClSize*, and α. Note that *MinClSize* and *MinPts* are typically set to the same value. Additionally, *RandMECThresh* and *RandDetThresh* are percentages which will often be set the same. Also, $\alpha \in [0,1]$ and is relatively easy to understand and set. The *NumSLExpand* parameter may be the most difficult to set. We recommend simply setting *NumSLExpand* = *MinPts* (as detailed previously in section 5.5).

It is important to note that HDBSCAN* and CASAR could be replaced with any density-based clustering algorithm in our semi-supervised density-based projected clustering method. Depending on the data set, other algorithms may perform better. In general, using HDBSCAN* to produce a dendrogram and extracting a single cluster from this dendrogram with CASAR will work best when individual clusters are embedded in unique subspaces, and when these clusters are relatively compact and small (compared to the size of the data set).

Complete pseudocode for our method follows:

Algorithm 1: A Semi-Supervised Density-Based Projected Clustering Method

Input: Collection of should-link object sets *collection_sl_sets*

1 *clustering_solution* = {}

2 For each attribute *a* in data set:

2.1 Construct a one-dimensional minimum spanning tree on *a* (see section 5.3)

3 For each should-link object set *sl_set* in *collection_sl_sets*:

3.1 For each attribute *a* in data set:

3.1.1 Compute a relevancy score for *sl_set* on *a* (see section 5.4)

3.2 Select an initial subspace *s* by comparing the relevancy scores of *sl_set* on attributes to a null distribution (see section 5.4)

3.3 Expand *sl_set* by finding objects with low mutual reachability distance to *sl_set* in the initial subspace *s* (see section 5.5)

3.4 Refine the subspace *s* using two statistical tests that use the determinant of the covariance matrix of *sl_set* (see section 5.6)

3.5 Run HDBSCAN* on the subspace *s* and use CASAR to extract the highest scoring cluster *c* (see section 4.1)

3.6 Add the subspace cluster (*c*,*s*) to *clustering_solution*

4 Return *clustering_solution*

Chapter 6: Evaluating Our Projected Clustering Method Experimentally

In this Chapter, we evaluate our method for semi-supervised density-based projected clustering. We compare our method to an existing semi-supervised projected clustering algorithm, SSPC [Yip et al., 2005 and Yip et al., 2009] and to an unsupervised projected clustering algorithm, P3C [Moise et al., 2006 and Moise et al., 2008]. P3C is used here as 'baseline' to explore the effectiveness of constraints. The implementation of SSPC we use is provided by the authors. The implementation of P3C we use comes from the ELKI framework [Schubert et al., 2015]. Note that this version of P3C does not implement section 3.5 of the P3C algorithm, where relevant attributes are assigned to found clusters. We still choose to use this version, for several reasons. Firstly, the attributes output by P3C are less interesting than the attributes output by SSPC and by our method. P3C's final set of attributes are assigned as a post-processing step, and as such, they provide less insight into the clustering process than the attributes output by SSPC and our method (which are assigned early in the clustering process). Secondly, P3C performs very poorly for most experiments in this Chapter, and as such, would often output either no attributes or a meaningless set of attributes.

Our comparisons are performed across a variety of data sets. We generate some entirely synthetic data sets, and also augment two real data sets with synthetic attributes. These real data sets are modified such that their clusters become embedded in subspaces, rather than having clusters which can be found in the full set of attributes. This strategy (adding noise attributes to real data sets) was employed by Guerra et al. [Guerra et al., 2014].

Throughout this Chapter, for the sake of brevity and clarity, we refer to our method as 'SSDBPC' (semi-supervised density-based projected clustering). This is not a name so much as it is a descriptor.

We use the *F-measure* as described by Banerjee et al. to measure the quality of clustering results (that is, the quality of selected objects) [Banerjee, et al., 2005]. This measure compares pairs of objects in the output clusters with pairs of objects in the true clusters. *Precision* is defined as the fraction of pairs of objects in the output clusters that are correctly paired. *Recall* is defined as the fraction of pairs of objects in the true clusters that appear in one or more output

clusters. The F-measure is the harmonic mean of precision and recall.

We use the *F_1 score* as described by Moise et al. to measure the quality of selected attributes [Moise et al., 2006 and Moise et al., 2008]. For each output cluster c_{a*}, we find the true cluster c_a in the data set which shares the most objects in common with c_{a*}. The *precision* of c_{a*} is defined as the number of attributes in common between c_{a*} and c_a divided by the number of attributes in c_{a*}. The *recall* of c_{a*} is defined as the number of attributes in common between c_{a*} and c_a divided by the number of attributes in c_a. The F_1 score for c_{a*} is the harmonic mean of precision and recall. The F_1 score for a set of output clusters is simply the mean average score of all clusters in the set.

6.1 Evaluation on Synthetic Data

As was the case when we compared CASAR and cluster stability, we construct synthetic data sets with a fixed set of features, and then vary up to one feature in each experiment. Note that, in some experiments, we vary an input parameter instead of a data set feature. Table 2 details the different features we vary when generating synthetic data. Default values for each feature are given in bold. We assign 100 objects to each cluster, and then assign any objects which do not have cluster membership to noise. Objects are allowed to have membership in multiple clusters, provided those clusters do not share any relevant attributes. This means that the number of noise objects will vary slightly from one data set to the next. For each data set, *n* (the number of objects) is equal to ((number of clusters)*(objects per cluster)*(1.0 + minimum percent noise objects). In a data set with 4 clusters, this means that *n* is 440.

The domain of all attributes is the range [0,1], and noise objects are distributed uniformly in this range. Additionally, cluster members are generated uniformly in the range [0,1] on

Data Set Feature	**Values (Default Bolded)**
number of clusters	2, 4, 6, 10 (**3**)
d (number of attributes)	25, 50, **100**, 150, 200
cluster minimum standard deviation	0.02, **0.04**, 0.06, 0.08, 0.1
objects per cluster	**100**
minimum percent noise objects	**0.1**
relevant attributes per cluster	**5**

Table 2: Features of the synthetic data sets that we generate.
Default values are displayed in bold.

irrelevant attributes. On relevant attributes, clusters are generated as multivariate Gaussians, with a maximum standard deviation in any relevant attribute 1.5x times larger than the given minimum standard deviation. As previously stated, clusters that have objects in common will not have any relevant attributes in common. Clusters that do not share any objects may overlap in some attributes. Additionally, we ensure that cluster centers (means) are at least 2 standard deviations away from from either limit of the [0,1] range in every attribute, and we do not allow cluster members to be generated outside of the [0,1] attribute range.

Each data point in our experiments is a mean over 10 iterations. Table 3 displays the input parameters we use when running SSDBPC, SSPC, and P3C. For each of these three algorithms, we experimented with a variety of different input parameters before deciding on the values given. These values gave good results (that is, the best results or nearly the best results) across a variety of different types of synthetic data sets. In each iteration, SSDBPC is run once with a fixed set of input parameters. SSPC and P3C are both run three times with varying values for *m* and *Poisson Threshold*, respectively. This is done because, for these data sets, SSDBPC produces reasonable results with a single set of parameters, whereas SSPC and P3C performed much better with varying parameters. For SSPC, we set *k* (the number of clusters) to the true number of clusters in the data set. We take the single best result (in terms of cluster quality, not attribute selection) for SSPC, and return that result for the iteration. The same is done for P3C. Note that, for SSDBPC, we set *MinPts*, *MinClSize*, and *NumSLExpand* to the same value, and we also set *RandMECThresh* and *RandDetThresh* to the same value (as we recommend users do). Additionally, note that the 'number of values in null distributions' parameter for SSDBPC refers to three different types of null distributions for three different statistical tests (as described in section 5.6), but we use the same size of null distribution for each test.

Algorithm	Parameter	Values (Default Bolded)
SSDBPC	number of values in null distributions	**1000**
SSDBPC	*MinPts = MinClSize = NumSLExpand*	**25**
SSDBPC	*RandMECThresh = RandDetThresh*	**0.001**
SSDBPC	α	**0.5**
SSPC	*m*	0.05, 0.15, 0.5
P3C	*Poisson Threshold*	1.0e-20, 1.0e-50, 1.0e-100
SSDBPC, SSPC	Number of should-link objects per set	2, **3**, 4, 6, 8, 10

Table 3: Algorithm parameters for SSDBPC, SSPC, and P3C.
Default values are displayed in bold.

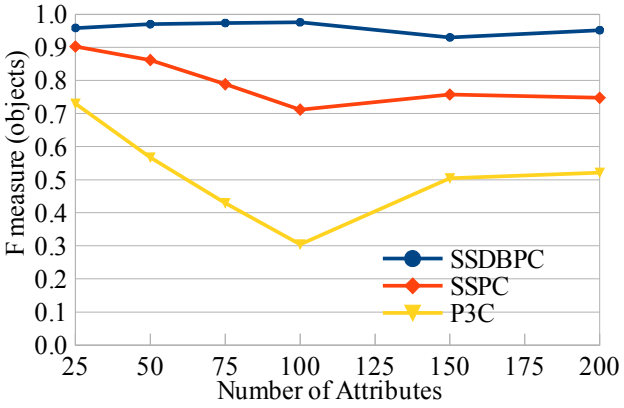

Fig 26a: Object Clustering Scores, Number of Clusters = 2

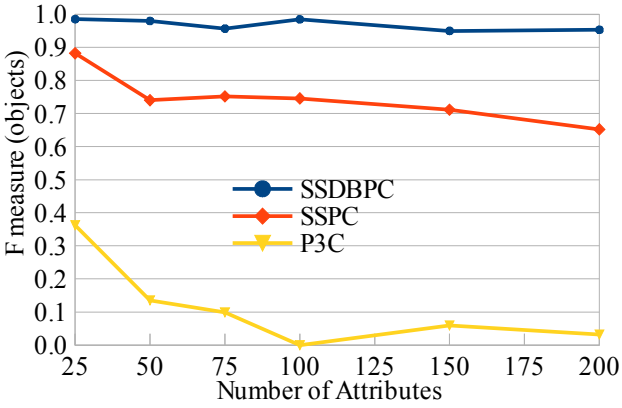

Fig 26b: Object Clustering Scores, Number of Clusters = 4

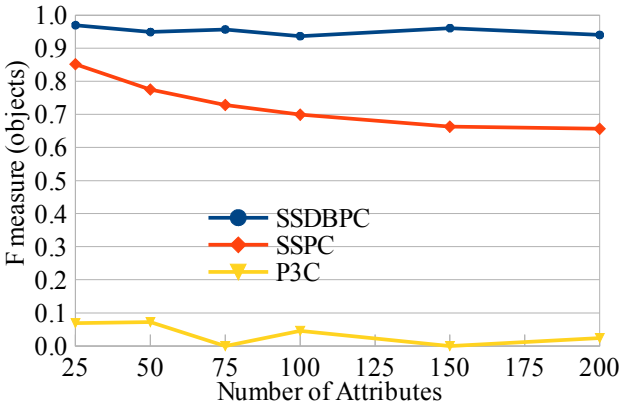

Fig 26c: Object Clustering Scores, Number of Clusters = 6

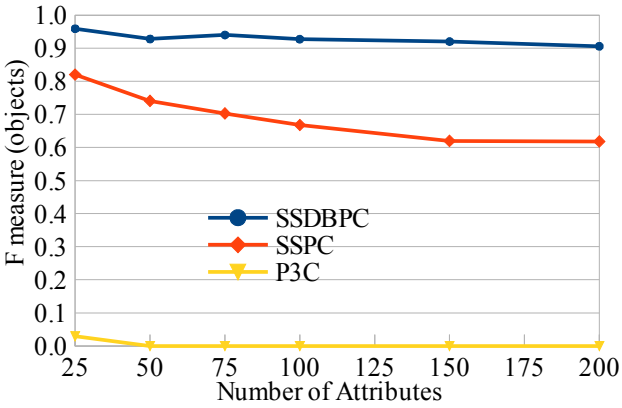

Fig 26d: Object Clustering Scores, Number of Clusters = 10

In each iteration of our experiments, a data set is randomly generated, and then results are obtained for each of the three algorithms on the data set. For each real cluster in the data set, we select a set of should-link objects which will be provided to SSDBPC and to SSPC. In order to ensure constraint consistency, each should-link object we select is a member of only one cluster. By default, we select 3 should-link objects per set (as detailed in table 3).

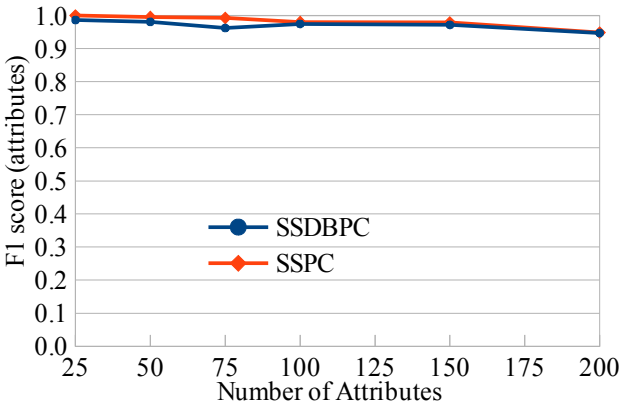

Fig 27a: Attribute Selection Scores, Number of Clusters = 4

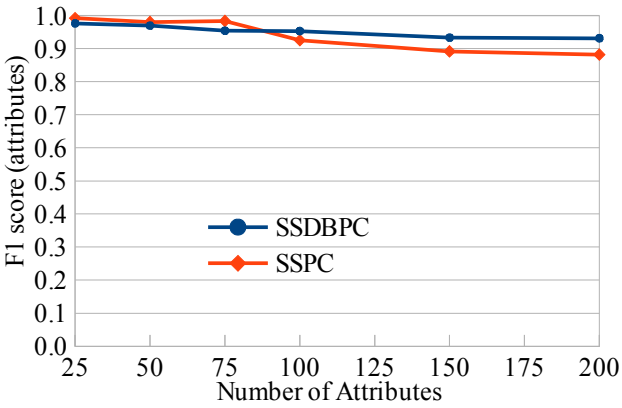

Fig 27b: Attribute Selection Scores, Number of Clusters = 10

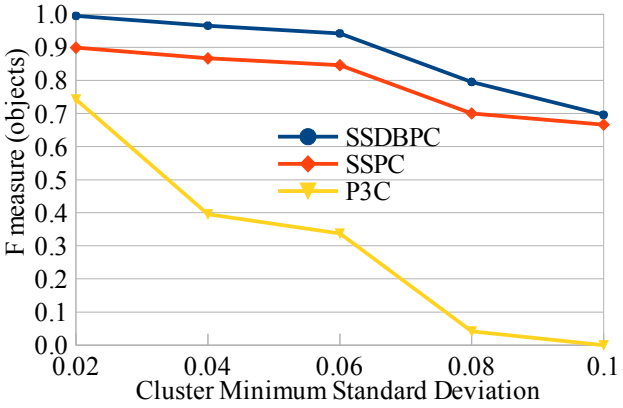

Fig 28a: Object Clustering Scores, Number of Attributes = 25

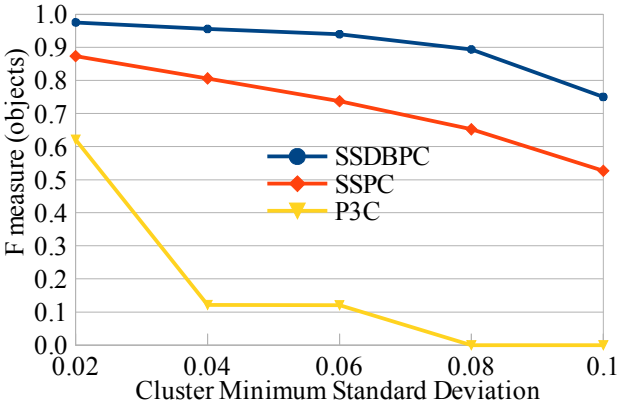

Fig 28b: Object Clustering Scores, Number of Attributes = 50

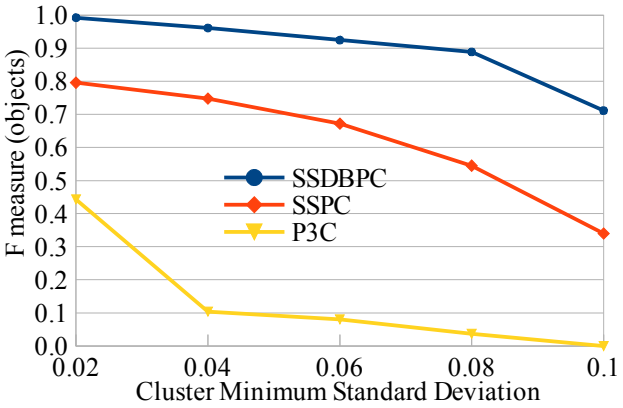

Fig 28c: Object Clustering Scores, Number of Attributes = 100

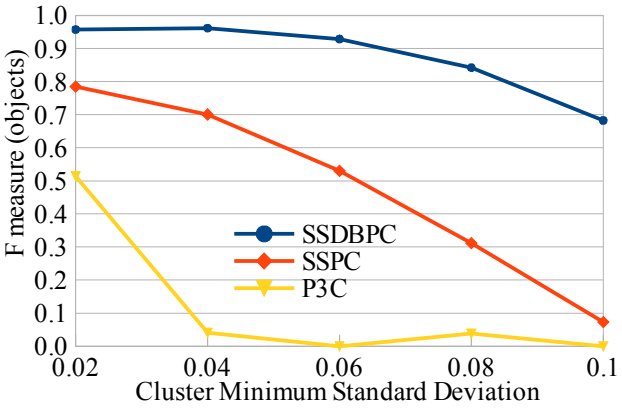

Fig 28d: Object Clustering Scores, Number of Attributes = 200

Our first set of experiments explore changing the number of clusters and number of attributes in the data set. Figures 26a through 26d display the differences in performance between SSDBPC, SSPC, and P3C in regards to clustering objects. The differences in performance between between SSDBPC and SSPC in regards to selecting attributes are displayed in Figures 27a and 27b. Both SSDBPC and SSPC are largely able to correctly identify relevant attributes. As the number of attributes in the data set increases, both algorithms have a slight decrease in performance. This occurs because the number of irrelevant attributes that are incorrectly selected, though very low, will continually increase as the number of irrelevant attributes increases. SSPC's performance in clustering objects degrades slightly more quickly than SSDBPC's performance as the number of attributes in the data set increases. SSPC's performance also suffers from slightly more degradation than SSDBPC's performance as the number of clusters in the data set increases. As the number of clusters in the data set increases, each cluster consists of a smaller percentage of the data set, and as such, becomes harder to detect. Due to this, P3C's performance degrades significantly as the number of clusters increases.

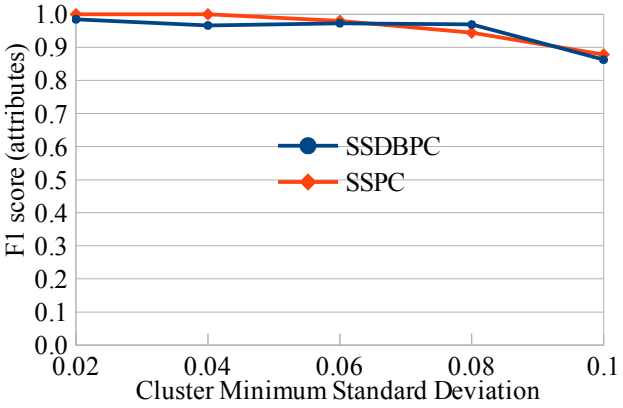

Fig 29a: Attribute Selection Scores,
Number of Attributes = 50

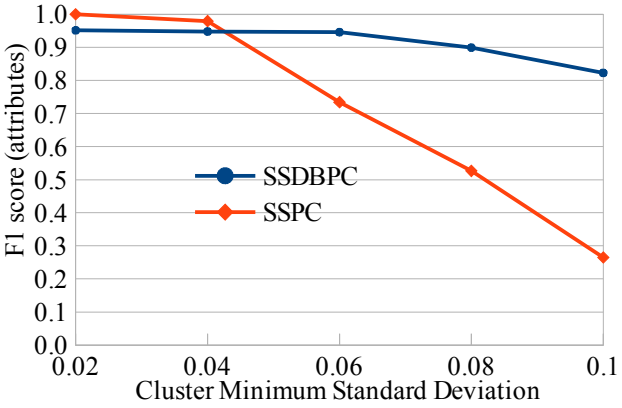

Fig 29b: Attribute Selection Scores,
Number of Attributes = 200

Obviously, P3C's inability to make use of should-link objects significantly impacts its performance.

In our second set of experiments, we vary the number of attributes in the data set and the cluster minimum standard deviation. Figures 28a through 28d display the performance of all three algorithms in regards to clustering objects. Figures 29a and 29b display the performance of

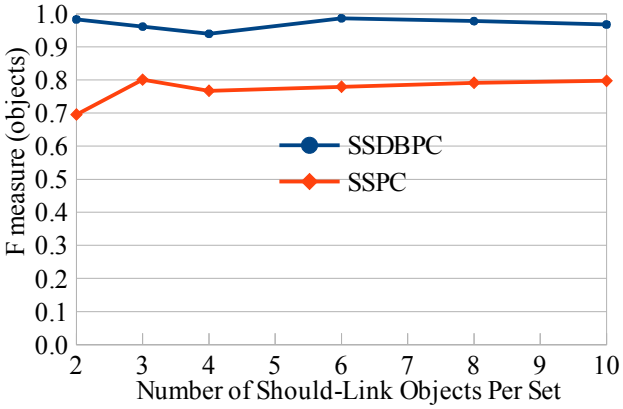

Fig 30a: Object Clustering Scores,
Minimum Cluster Standard Deviation = 0.02

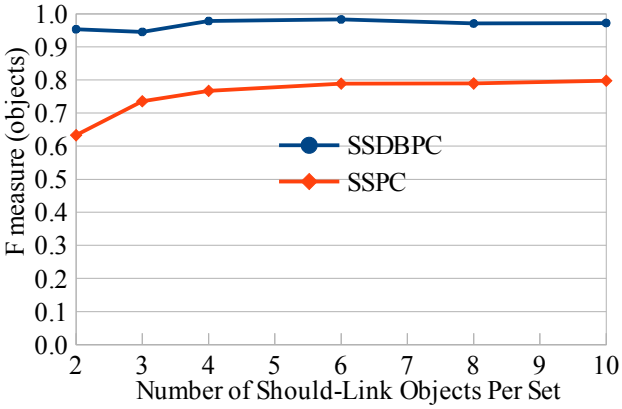

Fig 30b: Object Clustering Scores,
Minimum Cluster Standard Deviation = 0.04

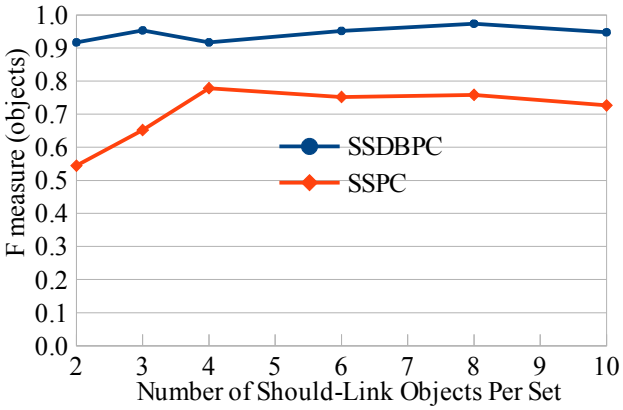

Fig 30c: Object Clustering Scores,
Minimum Cluster Standard Deviation = 0.06

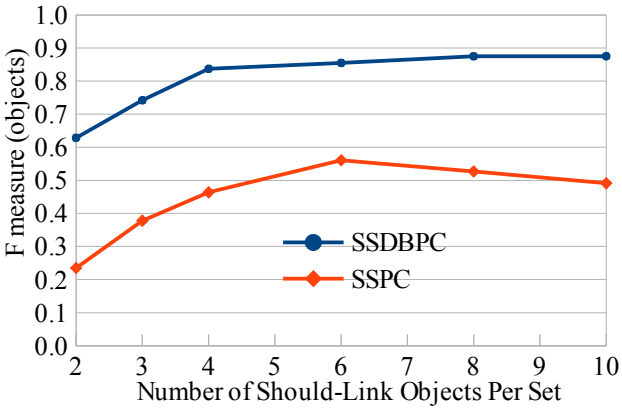

Fig 30d: Object Clustering Scores,
Minimum Cluster Standard Deviation = 0.1

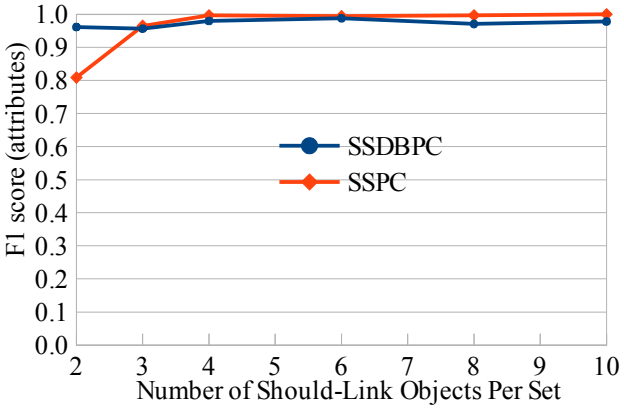

Fig 31a: Attribute Selection Scores,
Minimum Cluster Standard Deviation = 0.04

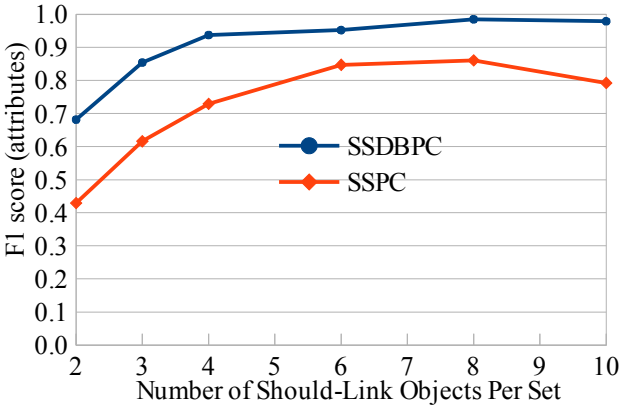

Fig 31b: Attribute Selection Scores,
Minimum Cluster Standard Deviation = 0.1

SSPC and SSDBPC in regards to selecting attributes. As expected, all three algorithms have decreasing performance as cluster compactness decreases. Once again, P3C's performance suffers the most, since the clusters become increasingly more difficult to detect as they become less compact. On data sets with a small number of attributes, SSPC generally matches SSDBPC's performance as cluster compactness decreases. On data sets with a larger number of attributes, SSPC suffers a much worse drop in performance than SSDBPC as cluster compactness decreases. Figure 29b shows that this performance drop is largely due to SSPC's inability to find the relevant subspace as clusters become increasingly less compact. The grid-based technique that SSPC uses to find attributes performs poorly in data sets that feature many irrelevant attributes and compact clusters. SSDBPC, on the other hand, is still able to detect both the relevant subspaces and the clusters in such data sets (albeit with decreased accuracy).

In our final set of experiments on synthetic data, we vary the number of should-link objects per set provided to SSPC and SSDBPC and the cluster minimum standard deviation. Figures 30a through 30d display the performance of SSPC and SSDBPC in regards to clustering objects, and Figures 31a and 31b display the performance of these algorithms in regards to selecting attributes. For both algorithms, providing additional should-link objects has little effect when the quality of clusters and attributes being found is already high. In cases where the quality of the result is poor, providing additional should-link objects has significant impact.

6.2 Evaluation on Real Data

In this section, we compare SSDBPC, SSPC, and P3C on modified versions of real data sets. Specifically, we augment the UCI machine learning repository [Lichman, 2013] data sets

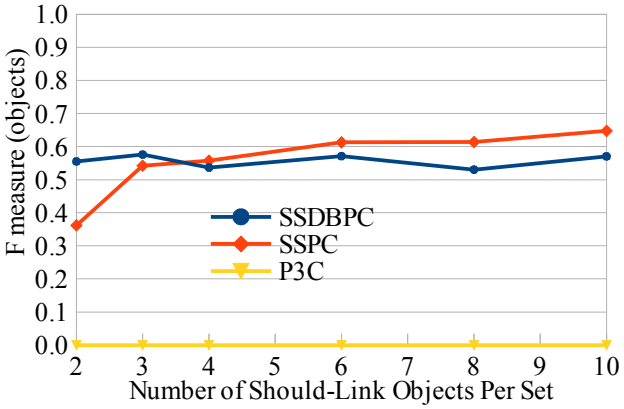

Fig 32a: Object Clustering Scores, Iris
Number of Noise Attributes = 25

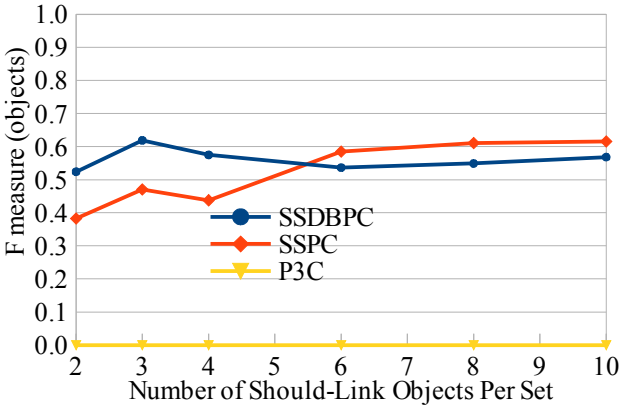

Fig 32b: Object Clustering Scores, Iris
Number of Noise Attributes = 50

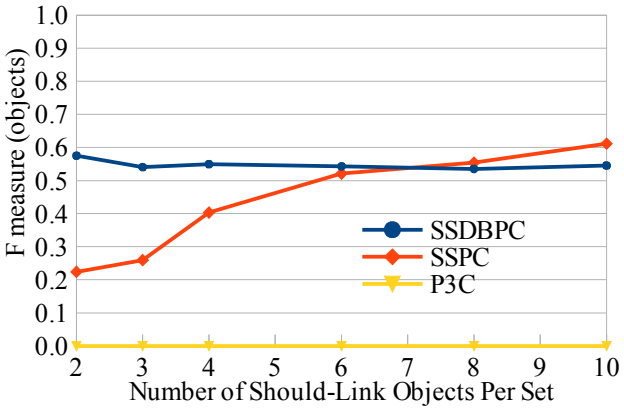

Fig 32c: Object Clustering Scores, Iris
Number of Noise Attributes = 100

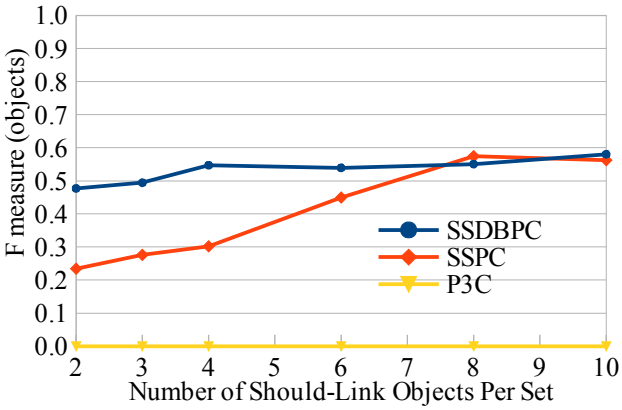

Fig 32d: Object Clustering Scores, Iris
Number of Noise Attributes = 200

'iris' and 'wine' by adding 'noisy attributes' to these data sets. The iris data set consists of 150 objects on 4 attributes, with 3 true clusters. Each cluster in the iris data set has 50 members. The wine data set consists of 178 objects on 13 attributes, with 3 true clusters. Clusters in the wine data set have 59, 71, and 48 members. For both data sets, we run all three algorithms using the parameter settings given in table 3. As was the case with synthetic data, each data point in our results is a mean average over 10 iterations. SSDBPC runs once for each iteration, while SSPC and P3C each run three times, with the best result saved for that iteration.

We modify these data sets by adding uniformly distributed noisy attributes. After these noisy attributes have been added, we normalize each attribute in the data set (including the new noisy attributes we have added). The attributes are normalized to a mean of 0.0 and a standard deviation of 0.5. Note that we could choose any arbitrary values for the mean and standard deviation to normalize to, provided every attribute is normalized to the same mean and standard deviation.

Figures 32a through 32d show the performance of all three algorithms in regards to

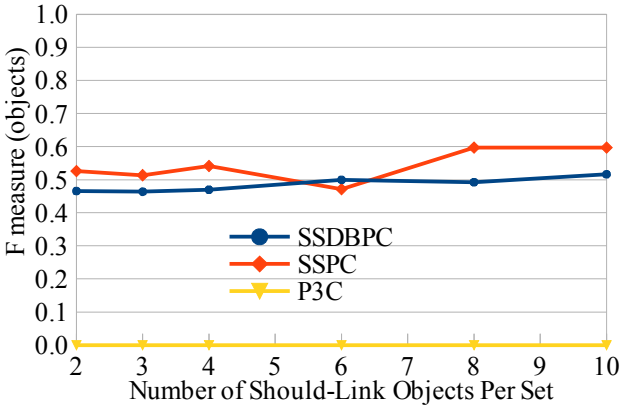

Fig 33a: Object Clustering Scores, Wine
Number of Noise Attributes = 25

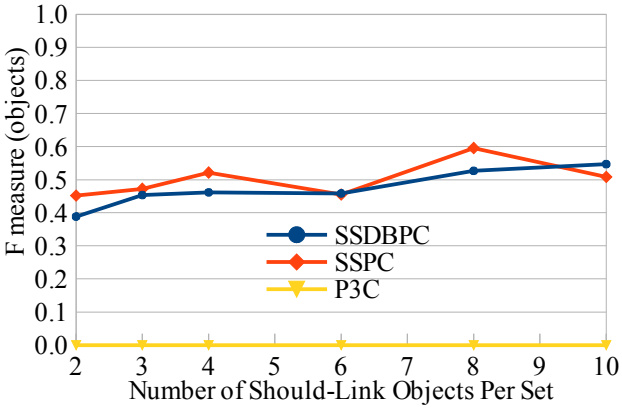

Fig 33b: Object Clustering Scores, Wine
Number of Noise Attributes = 50

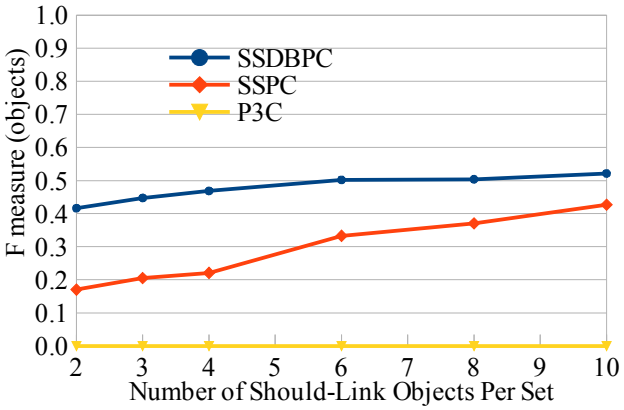

Fig 33c: Object Clustering Scores, Wine
Number of Noise Attributes = 100

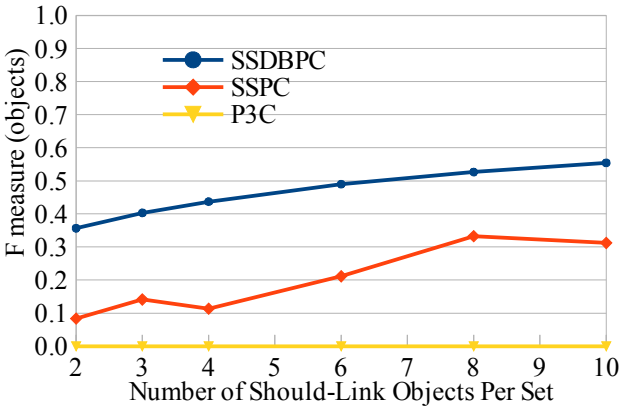

Fig 33d: Object Clustering Scores, Wine
Number of Noise Attributes = 200

clustering objects on the iris data set. Note that we vary the number of should-link objects per set provided to SSDBPC and SSPC. As was the case with synthetic data, SSDBPC handles an increasing number of attributes better than SSPC. Conversely, SSPC is better able to make use of an increased number of should-link objects. P3C was unable to find any clusters, regardless of the number of noise attributes.

Figures 33a through 33d display the performance of all three algorithms on the wine data set. For low numbers of noise attributes, SSPC is able outperform SSDBPC on the wine data set. As was the case with synthetic data and with the iris data set, SSDBPC outperforms SSPC as the number of attributes increases. Note that SSDBPC sees an improvement in performance as the number of should-link objects per set increases. P3C is once again unable to find any clusters.

Figures 34a and 34b display the average runtime of SSDBPC when compared to P3C. These runtimes are for a computer with an Intel Core i5-2500 CPU, 8 GB of RAM, and the Windows 7 Professional operating system. Note that we do not include results for SSPC: this is because the implementation of SSPC we were given produces enormous output files, such that a

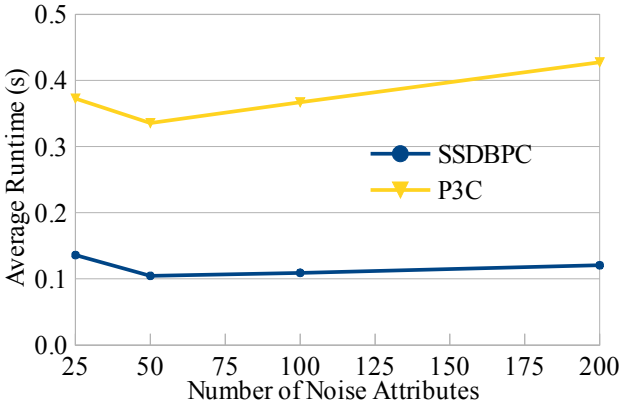

Fig 34a: Average Runtimes, Iris

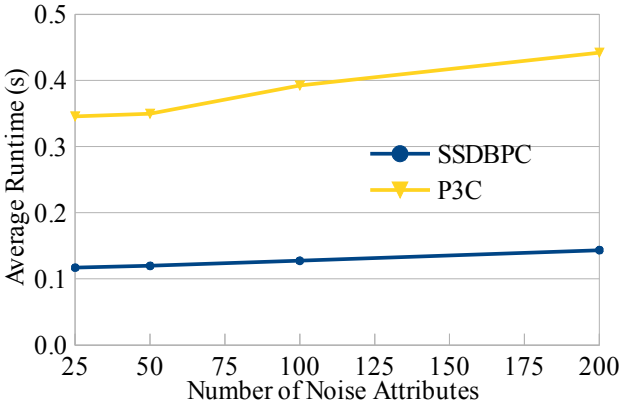

Fig 34b: Average Runtimes, Wine

large percentage of the runtime appears to be dominated by file I/O. Additionally, note that the runtime for SSDBPC does not include reading in the data set or writing the final labels to file, since these were handled in memory. P3C's runtime includes the time to start and end a separate process for P3C, as well as the time to read in the data set and write results to file, which undoubtedly causes some overhead. These results indicate that our method is extremely scalable in regards to the number of attributes in the data set.

6.3 Summary of Results

The experiments we have performed on synthetic and real data sets demonstrate that the greatest strength of our semi-supervised density-based projected clustering method is its ability to perform consistently as the number of noisy attributes in the data set increases. Our method's consistent performance is the result of the statistical tests we use, which can ensure a very low rate of false positives when selecting attributes for the relevant subspace. Additionally, our algorithm was able to outperform SSPC as the cluster minimum standard deviation increased (that is, as the clusters became increasingly less compact). These experiments also suggest that our method does not make use of constraints as well as SSPC – SSPC's performance generally improved more significantly than our method's performance as the number of should-link objects provided increased. Throughout these experiments, P3C struggled to provide meaningful clustering solutions. This suggests that certain types of subspace clusters may be extremely difficult to detect without the use of constraints.

Chapter 7: Conclusions

In this thesis, we provided density-based solutions to the problem of the 'curse of dimensionality'. Specifically, we presented a density-based clustering method known as CASAR and a density-based attribute selection technique. Both of these techniques are semi-supervised, and both are designed to aid in the process of finding density-based clusters which are embedded in subsets of the full set of attributes.

CASAR extracts a flat partition from an HDBSCAN* hierarchy in a manner very similar to FOSC. One measure that FOSC can use for cluster quality is cluster stability, which takes into account the longevity of clusters within an HDBSCAN* hierarchy. CASAR, on the other hand, measures cluster quality as the ratio between a cluster's compactness and separation, hence its name – Compactness And Separation Adjusted Ratio. CASAR's cluster quality measure more directly incorporates the density-based principle that clusters are regions of high density separated from each other by regions of low density. Additionally, CASAR is able to successfully extract a single cluster from surrounding noise, which cluster stability is unable to do. In a series of experiments, we showed that CASAR may not be superior to cluster stability in general, but does outperform cluster stability on data sets which feature a high percentage of noise objects, a low number of clusters, and clusters which are relatively compact. In general, when attempting to extract a density-based cluster from a subspace, we expect the cluster to consist of a relatively small percentage of compact objects which are embedded in a high percentage of noise. We argue that, given these results, CASAR should exceed at extracting density-based clusters from subspaces.

Having explained and experimentally tested CASAR, we went on to describe our method for semi-supervised density-based attribute selection. Our method takes as input a set of should-link objects that are known to belong to a single cluster, and outputs the relevant subspace for that cluster. We define subspace relevancy in terms of CASAR score. Our method computes an initial subspace using a statistical test that examines the minimum-eps-connectivity of should-link objects on individual attributes. Next, our method expands the set of should-link objects by examining the mutual reachability distance of objects in the initial subspace. This expanded set of should-link objects is then used to perform two statistical tests to refine the subspace via

covariance matrix determinants. Combining our method for attribute selection with CASAR results in a semi-supervised density-based projected clustering algorithm, which we experimentally compared against SSPC and P3C. In experiments on both synthetic data sets and augmented real data sets, P3C struggled to produce meaningful results, indicating that certain embedded clusters may be extremely difficult to detect without constraints. Our method suffered less of a performance decrease than SSPC as the number of noise attributes in the data set increased. Additionally, our method suffered less of a performance decrease as clusters became less compact. Both of these effects are due to the statistical tests we perform for attribute selection, which ensure a very low probability of mistakenly selecting an irrelevant attribute. In most cases, we also found that SSPC was better able to take advantage of an increase in the number of should-link objects. Conversely, this also indicates that our method operates well with even a small amount of labelled input.

7.1 Future Research

There are several ideas for future research which we believe may be worth pursuing. First of all, it would be interesting to construct a hybrid measure for cluster extraction from an HDBSCAN* hierarchy that makes use of both cluster stability and CASAR. While cluster stability seems to be more useful in general, it performs poorly on a variety of unique data sets (such as data sets that contain only a single cluster embedded in noise). CASAR performs well on these unique data sets. A simple hybrid measure could extract clusters separately using cluster stability and CASAR, and then use some validation technique to determine which result is better. Integrating these two methods more directly could potentially yield much better results.

Secondly, we would like to extend our method for semi-supervised density-based projected clustering to detect clusters for which no constraints have been provided. Obviously, this could trivially be completed by running an unsupervised projected clustering algorithm (such as P3C or STATPC) after running our method. It would be much more interesting (and likely lead to better clustering results) to find a way to use the information gained from the constraints to help guide the process of finding 'hidden' clusters. The set of clusters found using constraints might be able to provide 'clues' about the nature of the data set.

Finally, we believe it would be benefical to find additional uses (application domains, algorithms, etc) for some of the methodologies we have developed. For instance, being able to

construct one-dimensional minimum spanning trees in *O(n log n)* time might have additional applications in density-based clustering or in projected clustering. Our method for expanding the set of should-link objects (by finding objects with low mutual reachability distance to the should-link objects) might improve other semi-supervised methods. As well, our method for subspace refinement (which makes use of covariance matrix determinants) might have be relevant to other subspace clustering methods.

References

[Aggarwal et al., 1999] Aggarwal, C. C., Wolf, J. L., Yu, P. S., Procopiuc, C., & Park, J. S. (1999). Fast algorithms for projected clustering. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data (SIGMOD '99), 28*(2), (pp. 61-72). ACM.

[Aggarwal et al., 2001] Aggarwal, C. C., Hinneburg, A., & Keim, D. A. (2001). On the Surprising Behavior of Distance Metrics in High Dimensional Space. In *Database Theory – ICDT 2001,* (pp. 420-434). Springer Berlin Heidelberg.

[Agrawal et al., 1998] Agrawal R., Gehrke J., Gunopulos D., & Raghavan P. (1998). Automatic subspace clustering or high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data (SIGMOD '98)*, *27*(2), (pp. 94-105). ACM.

[Ankerst et al., 1999] Ankerst, M., Breunig, M. M., Kriegel, H. P., & Sander, J. (1999). OPTICS: ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM international conference on Management of data (SIGMOD '99), 28*(2), (pp. 49-60). ACM.

[Assent et al., 2007] Assent, I., Krieger, R., Muller, E., & Seidl, T. (2007). DUSC: Dimensionality unbiased subspace clustering. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007),* (pp. 409-414).

[Banerjee, et al., 2005] Banerjee, A., Krumpelman, C., Ghosh, J., Basu, S., & Mooney, R. J. (2005). Model-based overlapping clustering. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining (KDD '05)*. (pp. 532-537). ACM.

[Basu et al., 2002] Basu, S., Banerjee, A., & Mooney, R. (2002). Semi-supervised clustering by Seeding. In *Proceedings of the 19th International Conference on Machine Learning (ICML – 2002),* (pp. 27-24).

[Beyer et al., 1999] Beyer, K., Goldstein, J., Ramakrishnan, R., & Shaft, U. (1999). When is "nearest neighbor" meaningful?. In *Database Theory – ICDT'99: 7th International Conference,* (pp. 217-235). Springer Berlin Heidelberg.

[Böhm & Plant, 2008] Böhm, C., & Plant, C. HISSCLU: a hierarchical density-based method for semi-supervised clustering. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology (EDBT '08),* (pp. 440-451). ACM.

[Campello et al., 2013a] Campello, R. J. G. B., Moulavi, D., Zimek, A., & Sander, J. (2013). A framework for semi-supervised and unsupervised optimal extraction of clusters from hierarchies. In *Data Mining and Knowledge Discovery*, *27*(3), (pp. 344-371).

[Campello et al., 2013b] Campello, R. J. G. B., Moulavi, D., & Sander, J. (2013). Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining: 17th Pacific-Asia Conference, PAKDD 2013, Proceedings, Part 2,* (pp. 160-172). Springer Berlin Heidelberg.

[Campello et al., 2015] Campello, R. J. G. B., Moulavi, D., Zimek, A., & Sander, J. (2015). Hierarchical Density Estimates for Data Clustering, Visualization, and Outlier Detection. In *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *10*(1), 5. ACM.

[Cevikalp et al., 2008] Cevikalp, H., Verbeek, J., Jurie, F., & Klaser, A. (2008). Semi-supervised dimensionality reduction using pairwise equivalence constraints. In *VISAPP '08 – 3rd International Conference on Computer Vision Theory and Applications*, *1*, (pp. 489-496). INSTICC.

[Ester et al., 1996] Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (Kdd '96)*, (34), (pp. 226-231). AAAI Press.

[Fayyad et al., 1996] Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). The KDD process for extracting useful knowledge from volumes of data. In *Communications of the ACM*, *39*(11), (pp. 27-34). ACM.

[Fromont et al., 2009] Fromont, E., Prado, A., & Robardet, C. (2009). Constraint-based subspace clustering. In *Proceedings of the 2009 SIAM International Conference on Data Mining* (pp. 26-37). Society for Industrial and Applied Mathematics.

[Gajawafa & Toshniwal, 2012] Gajawada, S., & Toshniwal, D. (2012). Vinayaka: A semi-supervised projected clustering method using differential evolution. *International Journal of Software Engineering & Applications*, *3*(4). (pp. 77-85).

[Grira et al., 2004] Grira, N., Crucianu, M., & Boujemaa, N. (2004). Unsupervised and Semi-Supervised Clustering: a Brief Survey. '*A Review of Machine Learning Techniques for Processing Multimedia Content'*, *Report of the MUSCLE European Network of Excellence*.

[Guerra et al., 2014] Guerra, L., Bielza, C., Robles, V., & Larrañaga, P. (2014). Semi-supervised projected model-based clustering. *Data Mining and Knowledge Discovery*, *28*(4), (pp. 882-917). Springer US.

[Handl & Knowles, 2006]. Handl, J., & Knowles, J. (2006). Semi-supervised feature selection via multiobjective optimization. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings (IJCNN '06)*, (pp. 3319-3326). IEEE.

[Hartigan, 1975] Hartigan, J. A. (1975). *Clustering algorithms*. John Wiley & Sons, Inc..

[Hinneburg & Keim, 1998] Hinneburg, A., & Keim, D. A. (1998). An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of the 4th ACM International Conference on Knowledge Discovery and Data Mning (KDD '98),* (pp. 58-65). ACM.

[Hinneburg et al., 2000] Hinneburg, A., Aggarwal, C. C., & Keim, D. A. (2000). What is the nearest neighbor in high dimensional spaces?. In *Proceedings of the 26th International Conference on Very Large Databases,* (pp. 506-515).

[Hubert & Arabie, 1985] Hubert, L., & Arabie, P. (1985). Comparing partitions. In *Journal of Classification*, *2*(1), (pp. 193-218). Springer-Verlag.

[Jain et al., 1999] Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. In *ACM computing surveys (CSUR)*, *31*(3), (pp. 264-323). ACM.

[Kailing et al., 2004] Kailing, K., Kriegel, H. P., & Kröger, P. (2004). Density-Connected Subspace Clustering for High-Dimensional Data. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, (pp. 246-256). Society for Industrial and Applied Mathematics.

[Klein et al., 2002] Klein, D., Kamvar, S. D., & Manning, C. D. (2002). From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In *Proceedings of the 19th International Conference on Machine Learning (ICML '02),* (pp. 307-313).

[Lelis & Sander, 2009] Lelis, L., & Sander, J. (2009). Semi-supervised density-based clustering.

In *2009 Ninth IEEE International Conference on Data Mining (ICDM '09),* (pp. 842-847). IEEE.

[Lichman, 2013] Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[Moise et al., 2006] Moise, G., Sander, J., & Ester, M. (2006). P3C: A Robust Projected Clustering Algorithm. In *Sixth International Conference on Data Mining (ICDM '06)*, (pp. 414-425). IEEE.

[Moise et al., 2008] Moise, G., Sander, J., & Ester, M. (2008). Robust projected clustering. *Knowledge and Information Systems*, *14*(3), (pp. 273-298). Springer-Verlag.

[Moise & Sander, 2008] Moise, G., & Sander, J. (2008). Finding non-redundant, statistically significant regions in high dimensional data: a novel approach to projected and subspace clustering. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge discovery and data mining (KDD '08),* (pp. 533-541). ACM.

[Moise et al., 2009] Moise, G., Zimek, A., Kröger, P., Kriegel, H. P., & Sander, J. (2009). Subspace and projected clustering: experimental evaluation and analysis. In *Knowledge and Information Systems*, *21*(3), (pp. 299-326). Springer-Verlag.

[Moulavi et al., 2014] Moulavi, D., Jaskowiak, P. A., Campello, R. J. G. B., Zimek, A., & Sander, J. (2014). Density-based clustering validation. In *Proceedings of the 2014 SIAM International Conference on Data Mining,* (pp. 839-847). Society for Industrial and Applied Mathematics.

[Omran et al., 2007] Omran, M. G., Engelbrecht, A. P., & Salman, A. (2007). An overview of clustering methods. In *Intelligent Data Analysis*, *11*(6), (pp. 583-605). IOS Press.

[Prim, R. C., 1957] Prim, R. C. (1957). Shortest connection networks and some generalizations. In *Bell System Technical Journal*, *36*(6), (pp. 1389-1401). Blackwell Publishing Ltd.

[Procopiuc et al., 2002] Procopiuc, C. M., Jones, M., Agarwal, P. K., & Murali, T. M. (2002). A Monte Carlo algorithm for fast projective clustering. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data (SIGMOD '02)*, (pp. 418-427). ACM.

[Rand, W. M., 1971] Rand, W. M. (1971). Objective Criteria for the Evaluation of Clustering Methods. In *Journal of the American Statistical Association*, *66*(336), (pp. 846-850).

American Statistical Association.

[Ruiz et al., 2007] Ruiz, C., Spiliopoulou, M., & Menasalvas, E. (2007). C-DBSCAN: Density-Based Clustering with Constraints. In *Proceedings of the 11th International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, (pp. 216-223). Springer Berlin Heidelberg.

[Schubert et al., 2015] Schubert, E., Koos, A., Emrich, T., Züfle, A., Schmid, K. A., & Zimek, A. (2015). A framework for clustering uncertain data. *Proceedings of the VLDB Endowment*, *8*(12), (pp. 1976–1979). ACM.

[Tang et al., 2007] Tang, W., Xiong, H., Zhong, S., & Wu, J. (2007). Enhancing semi-supervised clustering: a feature projection perspective. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '07),* (pp. 707-716). ACM.

[Wagstaff et al., 2001] Wagstaff, K., Cardie, C., Rogers, S., & Schrödl, S. (2001). Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01),* (pp. 577-584). ACM

[Wagstaff et al., 2006] Wagstaff, K. L., Basu, S., & Davidson, I. (2006). When is constrained clustering beneficial, and why?. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence (AAAI '06)*.

[Xing et al., 2002] Xing, E. P., Jordan, M. I., Russell, S., & Ng, A. Y. (2002). Distance metric learning with application to clustering with side-information. In *Proceedings of the 15th International Conference on Neural Information Systems (NIPS '02),* (pp. 521-528). MIT Press.

[Yan & Domeniconi, 2006] Yan, B., & Domeniconi, C. (2006). Subspace metric ensembles for Semi-supervised Clustering of High Dimensional Data. In *Proceedings of the 17th European Conference on Machine Learning (ECML 2006),* (pp. 509-520). Springer Berlin Heidelberg.

[Yip et al., 2004] Yip, K. Y., Cheung, D. W., & Ng, M. K. (2004). Harp: A practical projected clustering algorithm. In *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, *16*(11), (pp. 1387-1397). IEEE.

[Yip et al., 2005] Yip, K. P., Cheung, D. W., & Ng, M. K. (2005). On discovery of extremely low-

dimensional clusters using semi-supervised projected clustering. In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*, (pp. 329-340). IEEE.

[Yip et al., 2009] Yip, K. Y., Cheung, L., Cheung, D. W., Jing, L., & Ng, M. K. (2009). A semi-supervised approach to projected clustering with applications to microarray data. *International journal of Data Mining and Bioinformatics*, *3*(3), (pp. 229-259).

[Zhang et al., 2007] Zhang, D., Zhou, Z. H., & Chen, S. (2007). Semi-Supervised Dimensionality Reduction. In *Proceedings of the 2007 SIAM International Conference on Data Mining,* (pp. 629-634). Society for Industrial and Applied Mathematics.

[Zhang et al., 2011] Zhang, X., Qiu, Y., & Wu, Y. (2011). Exploiting constraint inconsistence for dimension selection in subspace clustering: A semi-supervised approach. *Neurocomputing*, *74*(17), (pp. 3598-3608).