# Pinch Ratio Clustering from a Topologically Intrinsic Lexicographic Ordering

Douglas Heisterkamp[*] Jesse Johnson[†]

**Abstract**

This paper introduces an algorithm for determining data clusters called TILO/PRC (Topologically Intrinsic Lexicographic Ordering/Pinch Ratio Clustering). The theoretical foundation for this algorithm, developed in [14], uses ideas from topology (particularly knot theory) suggesting that it should be very flexible and robust with respect to noise.

The TILO portion of the algorithm progressively improves a linear ordering of the points in a data set until the ordering satisfies a topological condition called strongly irreducible. The PRC algorithm then divides the data set based on this ordering and a heuristic metric called the pinch ratio.

We demonstrate the effectiveness of TILO/PRC for finding clusters in a wide variety of real and synthetic data sets and compare the results to existing clustering methods. Moreover, because the output of TILO depends on the initial ordering, we consider the effects of different random orderings on the final clusters defined by PRC, and show that choosing an initial ordering based on a different clustering algorithm can improve the final clusters. These results verify that both the theoretical foundations of TILO and the heuristic notion of pinch ratio are reasonable.

## 1 Introduction

Data mining is the search for patterns and structure in large sets of (often high dimensional) data. This data is generally in the form of vectors, which one thinks of as points sampled from some underlying probability measure. This probability measure may be a sum of probability measures corresponding to different types of points, in which case one expects the different types of points to form geometrically distinguishable *clusters*.

Clustering algorithms fall into a number of categories determined by the assumptions they make about the underlying probability measures and their approach to searching for clusters [6, 13]. The $K$-means algorithm [10] assumes that the underlying measures are Gaussian distributions centered at $K$ points throughout a Euclidean space. Hierarchical clustering algorithms [3, 22] arrange the data points by building a tree in either a top-down or bottom up manner. This allows much more flexibility of the model and the metric, but for many of these algorithms, the final structure is dependent on the order in which the tree is constructed.

Graph partitioning algorithms translate the data points into a graph with weighted edges in which the weights reflect the similarity between points in whatever metric is most natural for the given type of data. The clusters are defined by subgraphs that can be separated from the whole graph by removing relatively few edges. Graph clusters should come very close to realizing the Cheeger constant for the graph and there are algorithms for finding them based on linear programming [17] as well as spectral analysis of the Laplacian of the adjacency matrix [4]. Carlsson and Memoli [1] recently introduced a hierarchical clustering method based on encoding the data as a simplicial complex (a generalization of a graph), giving the algorithm a very strong grounding in topology.

In [14] another topological approach to graph clustering was suggested, based on the idea of *thin position* for knots and 3-manifolds [9, 20]. In the context of 3-manifolds, thin position determines minimal genus Heegaard splittings, which are related to minimal surfaces [19] and the Cheeger constant [15]. In [14] thin position was translated into the framework of graph partitioning/clustering. The present paper describes and studies the resulting algorithm that we will call TILO (for Topologically Intrinsic Lexicographic Ordering). TILO makes few assumptions about the underlying structure of the data. For example it does not make the assumption that the clusters are of equal size or the assumption of similar cluster shapes. It also does not need to know the desired number of clusters.

TILO returns a linear ordering of the vertices of a graph in which it considers as strongly irreducible (see below in Theorem 2.1). A linear *ordering* $\boldsymbol{o}$ is a permutation of the integer indices of the vertices of a graph. If a graph contains vertices $\boldsymbol{v}_i$ for $0 \leq i < N$ then the $i$th vertex of order $\boldsymbol{o}$ is $\boldsymbol{v}_{\boldsymbol{o}(i)}$ . TILO guarantees that all of the local minima associated with the ordering are valid locations to split the graph into pinch clusters (see below in Definition 2.1). The ordering that TILO returns may have a large number of local minima, some reflecting the structure of the data and other resulting from noise. To address the question of selecting from the valid locations found by TILO to create the desired number of clusters, we created the pinch ratio clustering algorithm (PRC).

A road map of the paper is as follows. In section 2, the mathematical foundation from [14] is summarized

[*]doug@cs.okstate.edu, Department of Computer Science, Oklahoma State University.

[†]jjohnson@math.okstate.edu, Department of Mathematics, Oklahoma State University.

and the TILO algorithm is presented. In section 3, we propose a new measure called *pinch ratio* which is based on concept of a topological thin position and thick width. We then present the pinch ratio clustering algorithm (PRC). In section 4, we evaluate TILO and PRC on real world data sets and in comparison to other common clustering algorithms. The appendix contains the experimental parameters.

## 2 Topologically Intrinsic Lexicographic Ordering

The TILO algorithm was inspired by a technique called *thin position* that has been used in knot theory and 3-dimensional topology [9, 20]. Thin position has been shown to be related to Cheeger constants [15] and to minimal surfaces [19]. The TILO algorithm uses a linear ordering of the vertices of a graph to induce a "width" and then looks for ways to modify the order to find "thinner" orderings.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where $\mathcal{V}$ is the set of $N$ vertices and $\mathcal{E}$ the set of (weighted or unweighted) edges. We will assume that each edge has distinct endpoints. For a vertex subset $\mathcal{A} \subset \mathcal{V}$, the denote the complement as $\bar{\mathcal{A}} = \mathcal{V} \setminus \mathcal{A}$. For a vertex subset $\mathcal{A} \subset \mathcal{V}$, the *boundary* $\partial\mathcal{A} \subset \mathcal{E}$ is the set of edges with one endpoint in $\mathcal{A}$ and the other endpoint in the complement $\bar{\mathcal{A}}$. The *size* of the boundary $|\partial\mathcal{A}|$ is the sum of the edge weights. A weight value of one is used for a graph with unweighted edges. There is no universally agreed upon definition of a cluster, but roughly speaking one would want it to have a relatively small boundary relative to the weights of the edges that do not cross its boundary. We will use the following definition from [14]:

Definition 2.1. *A pinch cluster is a set of vertices $\mathcal{A} \subset \mathcal{V}$ with the property that for any sequence of vertices $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m$, if adding $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m$ to $\mathcal{A}$ or removing $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m$ from $\mathcal{A}$ creates a set with smaller boundary then for some $k < m$, adding/removing $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k$ to/from $\mathcal{A}$ creates a set with strictly larger boundary.*

In other words, if you add a sequence of vertices to $\mathcal{A}$ or remove a sequence of vertices from $\mathcal{A}$ one at a time, the boundary size must increase before it decreases.

Let ordering $\boldsymbol{o}$ be a permutation of the integers zero to $N$-1. Define the set of vertices $\mathcal{A}_i$ with respect to ordering $\boldsymbol{o}$ and vertices $\mathcal{V}=\{\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{N-1}\}$ as $\mathcal{A}_i = \{\boldsymbol{v}_{\boldsymbol{o}(k)} \mid 0 \le k \le i\}$. That is, the vertices using the first $i+1$ indices from ordering $\boldsymbol{o}$. Let $\boldsymbol{b}=[|\partial\mathcal{A}_0|, \ldots, |\partial\mathcal{A}_{n-1}|]$ be the array of boundaries of $\mathcal{A}_i$ with respect to ordering $\boldsymbol{o}$ and graph $\mathcal{G}$ for $i$ ranging from zero to $N$-1. Note that $\boldsymbol{b}_{n-1} = 0$ as the boundary between the whole set and the empty set is zero. For convenience, we will logically extend $\boldsymbol{b}$ with $\boldsymbol{b}_{-1} = 0$.

```
TILO (graph $\mathcal{G}$, order $\boldsymbol{o}$)
1. repeat
2.    flag = false
3.    calculate boundaries using current ordering $\boldsymbol{o}$
4.    for each max flat $f$ with range ($b$,$c$) do
5.       $a$ = end index of previous local min flat
6.       $d$ = start index of next local min flat
7.       v = max $\{0\} \cup$
           $\{s_{b,k} - s_{b,b+1} - 2\boldsymbol{G}[\boldsymbol{o}(k), \boldsymbol{o}(b+1)] \mid a \le k \le b,\ s_{b,k} > 0\}$
           $\cup$
           $\{-s_{c,k} + s_{c,c} - 2\boldsymbol{G}[\boldsymbol{o}(k), \boldsymbol{o}(c)] \mid c < k \le d,\ s_{c,k} < 0\}$
8.       if $v > 0$ then
9.          let $k$ be an index with value $v$
10.         if $k \le b$ then
11.            cyclic shift $k$ to $b+1$ in order $\boldsymbol{o}$
12.         else
13.            cyclic shift $k$ to $c$ in order $\boldsymbol{o}$
14.         end-if-else
15.         flag = true
16.      end-if
17.   end-for
18. until not flag
19. return order $\boldsymbol{o}$
```

Figure 1: The Topologically Intrinsic Lexicographic Ordering Algorithm

The *width* of the ordering $\boldsymbol{o}$ is the values of array $\boldsymbol{b}$ sorted into non-increasing order. We will compare the widths of different orderings using lexicographic (dictionary) ordering: Given widths $\boldsymbol{u}$ and $\boldsymbol{v}$ then $\boldsymbol{u} < \boldsymbol{v}$ if there is a value $i$ such that $\boldsymbol{u}_i < \boldsymbol{v}_i$ and $\boldsymbol{u}_j = \boldsymbol{v}_j$ for every $j < i$. In other words, start from the front of the arrays and find the first element in which they disagree and make the comparison based on that element.

The TILO algorithm is presented in Figure 1. The algorithm uses a family of permutations called *shifts* to reduce the width of an initial ordering. When the width can no longer be reduced, the ordering is called *strongly irreducible* and the local minimums in the boundary array indicate locations where the vertices can be partitioned into two pinch clusters. To discuss the algorithm, we need to define the additional terms, *slope*, *flat*, and *shift*.

The *slope* $s_{\mathcal{A}}(v)$ of a vertex $\boldsymbol{v} \in \mathcal{V}$ with respect to subset $\mathcal{A} \subset \mathcal{V}$ is the sum of the edge weights from $\boldsymbol{v}$ to vertices in $\bar{\mathcal{A}}$ minus the sum of the edge weights from $\boldsymbol{v}$ to vertices in $\mathcal{A}$. If $\boldsymbol{v} \notin \mathcal{A}$ then this is the amount that the boundary $|\partial\mathcal{A}|$ will increase if we add $\boldsymbol{v}$ into the set. If $\boldsymbol{v} \in \mathcal{A}$, this is the amount $|\partial\mathcal{A}|$ will decrease if we remove it from $\mathcal{A}$. Since $\mathcal{A}_i = \mathcal{A}_{i-1} \cup \{\boldsymbol{v}_{\boldsymbol{o}(i)}\}$

then $|\partial \mathcal{A}_i| = |\partial \mathcal{A}_{i-1}| + s_{\mathcal{A}_{i-1}}(\boldsymbol{v}_{\boldsymbol{o}(i)})$. We will use the abbreviation $s_{i,j}$ to mean $s_{\mathcal{A}_i}(\boldsymbol{v}_{\boldsymbol{o}(j)})$.

A *flat* in $\boldsymbol{b}$ is an interval $[i,j]$ with $i \leq j$ such that $\boldsymbol{b}_i = \boldsymbol{b}_{i+1} = \cdots = \boldsymbol{b}_j$, $\boldsymbol{b}_{i-1} \neq \boldsymbol{b}_i$, and $\boldsymbol{b}_{j+1} \neq \boldsymbol{b}_j$. Note that a flat may consist of a single element $[i,i]$. A flat is *locally maximal* if $\boldsymbol{b}_{i-1} < \boldsymbol{b}_i$ and $\boldsymbol{b}_{j+1} < \boldsymbol{b}_j$. Similarly, a flat is *locally minimal* if $\boldsymbol{b}_{i-1} > \boldsymbol{b}_i$ and $\boldsymbol{b}_{j+1} > \boldsymbol{b}_j$. We will say that $i \in [0, N-1]$ is a *local minimum/maximum* if it is contained in a locally minimal/maximal (respectively) flat.

A cyclic *shift* on a subsequence of integers will move each integer to the left (or right) by one with wrapping around the ends of the subsequence. Given an initial sequence $\boldsymbol{x} = [\ldots, \boldsymbol{x}_{i-1}, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_{j-1}, \boldsymbol{x}_j, \boldsymbol{x}_{j+1}, \ldots]$ then cyclic shifting $i$ to $j$ moves all of the elements from $i$ to $j$ left one location and wraps $i$ around to $j$'s location: $\boldsymbol{y} = [\ldots, \boldsymbol{x}_{i-1}, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_{j-1}, \boldsymbol{x}_j, \boldsymbol{x}_i, \boldsymbol{x}_{j+1}, \ldots]$ and cyclic shifting $j$ to $i$ moves all of the elements from $i$ to $j$ right one location and wraps $j$ around to $i$'s location: $\boldsymbol{x} = [\ldots, \boldsymbol{x}_{i-1}, \boldsymbol{x}_j, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_{j-1}, \boldsymbol{x}_{j+1}, \ldots]$ where $\boldsymbol{x}_i$ is the integer value of the sequence at the $i$th location and $\boldsymbol{y}$ is the sequence resulting from the shift.

The TILO algorithm presented in Figure 1 can be summarized as calculate the boundary of the current ordering, find and apply the best valid shift for each maximum flat of the boundary, repeat until then are no more shifts. TILO uses Lemma 3 of [14] to determine if a shift is valid. This criterion is implemented in lines 7 and 8 of Figure 1. Verbally, this is saying that any location $k$ from the end of the previous local minimum flat to the start location $b$ of the current maximum flat is valid if slope $s_{b,k}$ and expression $s_{b,k} - s_{b,b+1} - 2\boldsymbol{G}[\boldsymbol{o}(k), \boldsymbol{o}(b+1)]$ are both greater than zero. Note that $\boldsymbol{G}[\boldsymbol{o}(k), \boldsymbol{o}(b+1)]$ means the weight of the edge between vertices $\boldsymbol{v}_{\boldsymbol{o}(k)}$ and $\boldsymbol{v}_{\boldsymbol{o}(b+1)}$. If $k$ is valid then cyclic shifting $k$ to $b+1$ in $\boldsymbol{o}$ will result in a new boundary array with all values less than or equal to the values in the current boundary array. The second part of the condition is saying that any location $k$ from the end location $c$ of the current maximum flat to the start of the next location minimum flat is a valid location if slope $s_{c,k}$ less than zero and expression $-s_{c,k} + s_{c,c} - 2\boldsymbol{G}[\boldsymbol{o}(k), \boldsymbol{o}(c)]$ is greater than zero. If $k$ is valid then cyclic shifting $k$ to $c$ will reduce the boundary array values. The location $k$ that is selected is the one with the largest expression value as that corresponds to the amount that the boundary at the maximum flat location will be reduced. Lemma 3 of [14] guarantees these shifts will strictly reduce the width of the ordering.

When TILO stops due to there being no more valid shifts, then the following theorem from [14] holds.

Theorem 2.1. *Given any initial ordering, if one repeatedly applies shifts that strictly reduce the width of the ordering, the process will terminate with a strongly irreducible ordering after a finite number of steps. In a strongly irreducible ordering, if $i$ is a local minimum with respect to width then $\mathcal{A}_i$ and its complement are pinch clusters.*

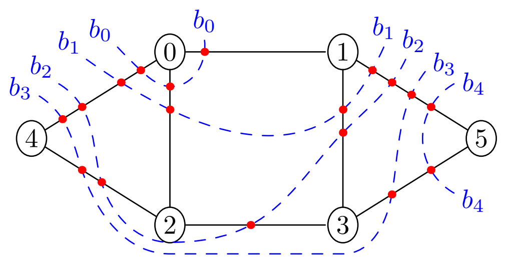
TILO is a greedy algorithm and is not guaranteed to find an ordering with the globally minimal width. But Theorem 2.1 guarantees that it will converged to an ordering with a locally minimal width and that all local minima in the corresponding boundary array are locations at which the vertices can be partitioned into a pair of pinch clusters. An interesting facet of TILO is how it is greedy. It does not greedily search orderings to create good local minimums. Instead, it reduces the local maxima first as they are the leading terms in the width of the ordering. After it finishes squeezing the local maxima, the local minima are also set. Normally, TILO stops as soon as the local minima and local maxima are fixed. By a slight extension of Lemma 3 of [14] TILO can run until the width at all locations is minimized. This can be useful if we want to use the width of an ordering to select between two different runs of TILO.

An example of running TILO on a small graph is provide in Figure 2. The graph has a slight preference for an even-odd clustering of the vertices. The initial ordering $\boldsymbol{o}$=$[0,1,2,3,4,5]$ has boundary $\boldsymbol{b}$=$[3,4,5,4,2]$ and width $\boldsymbol{w}$=$[5,4,4,3,2]$. After four shifts, the final ordering is $\boldsymbol{o}$=$[4,0,2,1,3,5]$ with boundary $\boldsymbol{b}$=$[2,3,2,3,2]$ and width $\boldsymbol{w}$=$[3,3,2,2,2]$. The local minimum at $\boldsymbol{b}_2$ corresponds to $\mathcal{A}_2 = \{\boldsymbol{v}_{\boldsymbol{o}(0)}, \boldsymbol{v}_{\boldsymbol{o}(1)}, \boldsymbol{v}_{\boldsymbol{o}(2)}\} = \{\boldsymbol{v}_4, \boldsymbol{v}_0, \boldsymbol{v}_2\}$ which is a pinch cluster. The complement, $\bar{\mathcal{A}}_2 = \{\boldsymbol{v}_1, \boldsymbol{v}_3, \boldsymbol{v}_5\}$ is also a pinch cluster.

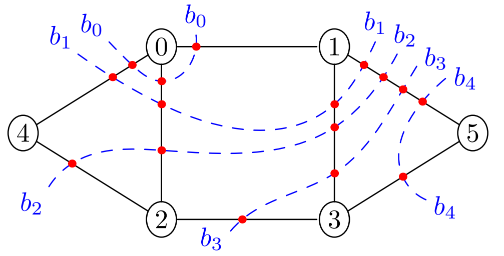
The internal loop of TILO is fast as its operations are simple. The boundary and the slopes $s_{i,j}$ can be pre-calculated and incrementally updated based on the range of the shifts. An additional note for step 5 of the TILO algorithm is needed as care must be taken to not have overlapping shifts. If $d$ was used in the reducing the previous maximum flat and the local minimum flat is just the single location $d$ then the next maximum flat would need to start its search at $a+1$, which would be $d+1$.

## 3 Pinch Ratio Clustering

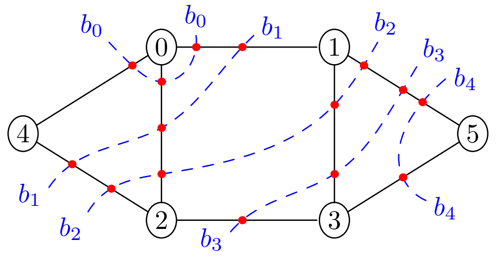
The boundary array associated with strongly irreducible ordering will have sets of local minima and local maxima. Any local minima can be selected to form a pinch cluster. Any measure of cluster quality may be used to make the selection, such as *normalized cut* [21]. Using our notation, the normalized cut value of the sets $\mathcal{A}_k$



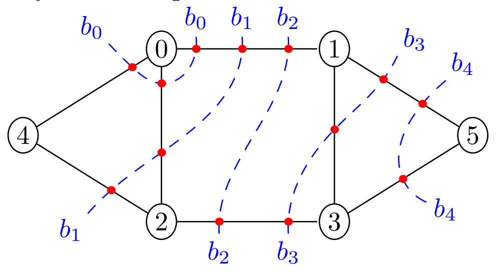
Initial order $\boldsymbol{o}$=[0, 1, 2, 3, 4, 5]; boundary $\boldsymbol{b}$=[3,4,5,4,2]. Cyclic shifting order index 4 to index 2.



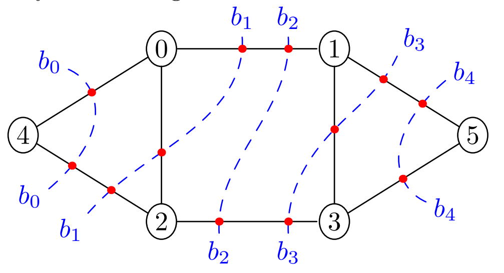
Order $\boldsymbol{o}$=[0, 1, 4, 2, 3, 5]; boundary $\boldsymbol{b}$=[3,4,4,3,2]. Cyclic shifting order index 1 to index 2.



Order $\boldsymbol{o}$=[0, 4, 1, 2, 3, 5]; boundary $\boldsymbol{b}$=[3,3,4,3,2]. Cyclic shifting order index 2 to index 3.



Order $\boldsymbol{o}$=[0, 4, 2, 1, 3, 5]; boundary $\boldsymbol{b}$=[3,3,2,3,2]. Cyclic shifting order index 0 to index 1.



Order $\boldsymbol{o}$=[4, 0, 2, 1, 3, 5]; boundary $\boldsymbol{b}$=[2,3,2,3,2].

Figure 2: TILO example on a graph with a slight preference for an even-odd clustering of the vertices.

and $\bar{\mathcal{A}}_k$ is

$$\textbf{NCut}_k = \frac{\boldsymbol{b}_k}{|\mathcal{A}_k|} + \frac{\boldsymbol{b}_k}{|\bar{\mathcal{A}}_k|}. \tag{3.1}$$

This normalizes the local cut value $\boldsymbol{b}_k$ by the volumes of the two partitions. We propose using the additional information available in array $\boldsymbol{b}$ when TILO finishes to create a measure that we call the *pinch ratio*. The pinch ratio will use the idea of a *thick width* which is the minimum over all possible ordering of the max cut of a set. The thick width give us a measure of the internal cohesiveness of a set. The max cut of a set is the largest boundary of a binary partitioning of the set. When restricted to an ordering, it is the largest value in the boundary array. TILO reduces this value. We have two thick widths, one for $\mathcal{A}_k$ and one for $\bar{\mathcal{A}}_k$. We use the smaller thick width in determining the pinch ratio. The pinch ratio at a location $k$ is the ratio of the thin position width over the thick width. In terms of the boundary, it is

$$\textbf{pinch ratio}_k = \frac{\boldsymbol{b}_k}{\min\left(\max\limits_{p\le i\le k} \boldsymbol{b}_i\,,\ \max\limits_{k<j\le q} \boldsymbol{b}_j\right)}. \tag{3.2}$$

where $p \le k < q$ is the range of interest in the boundary (typically, $p = 0$ and $q = N - 1$).

To motivate the idea, think of a rubber band stretched over the point set at linear position $k$ with the tension in the rubber band proportional to the cost of cutting the adjacency graph of the point set into sets $\mathcal{A}_k$ and $\bar{\mathcal{A}}_k$ (i.e., the boundary value $\boldsymbol{b}_k$). The rubber band could be pulled off by either going over set $\mathcal{A}_k$ or set $\bar{\mathcal{A}}_k$. Either set is traversed in the given linear order and the tension in the rubber band at each location is proportional to the boundary at that location. The thick width is the minimum maximum effort needed to remove the rubber band, i.e. the minimum of the maximum boundary from each set. Since TILO reduces the maximum boundary values until it reaches a strongly irreducible ordering, the maximum boundary of $\mathcal{A}_k$ can be viewed as an approximation to the minimum max cut of $\mathcal{A}_k$ with respect to the entire graph.

Pinch ratio clustering uses the pinch ratio to find a good location to split the current set with the range of the max operators in the denominator of (3.2) limited to the current set. The pinch ratio clustering algorithm PRC is presented in Figure 3. FindSplit($\mathcal{C}$,$\boldsymbol{o}$) in lines 4 and 15 of PRC returns the value and location of the best pinch ratio in the set $\mathcal{C}$ with respect to the order $\boldsymbol{o}$. If no split locations are found then infinity is returned for the value and minus one for the location. Lines 13 and 14 are not needed if three or fewer clusters are to be found.

```
PRC(graph $\mathcal{G}$, order $\boldsymbol{o}$, number of partitions $k$)
1.  run TILO on $\mathcal{G}$ and $\boldsymbol{o}$
2.  initialize $\boldsymbol{Q}$ to empty min priority queue
3.  $\mathcal{C} \leftarrow \{i | i \in \boldsymbol{o}\}$
4.  $q$,$t \leftarrow$ FindSplit($\mathcal{C}$,$\boldsymbol{o}$)
5.  enqueue tuple ($q$,$t$,$\mathcal{C}$) on to $\boldsymbol{Q}$
6.  while length of $\boldsymbol{Q} < k$ do
7.      $q_i$,$t_i$,$\mathcal{C}_i \leftarrow \boldsymbol{Q}$.dequeue()
8.      if no split location $t_i$ then
9.          break out of while loop
10.     end-if
11.     $\mathcal{C}_j$ ,$\mathcal{C}_k \leftarrow$ Split($\mathcal{C}_i$ ,$t_i$)
12.     for $y$ in $\{j, k\}$ do
13.         $\boldsymbol{p} \leftarrow$ MoveToFront($\mathcal{C}_y$ ,$\boldsymbol{o}$)
14.         run TILO on $\mathcal{G}$ and $\boldsymbol{p}$ restricted to $\mathcal{C}_y$
15.         $q_y$,$t_y \leftarrow$ FindSplit($\mathcal{C}_y$ ,$\boldsymbol{p}$)
16.         enqueue tuple ($q_y$,$t_y$,$\mathcal{C}_y$ ) on to $\boldsymbol{Q}$
17. end-while
18. return $\boldsymbol{Q}$     // each $\mathcal{C}_j$ in $\boldsymbol{Q}$ represents
                    // a different cluster
```

Figure 3: The Pinch Ratio Clustering Algorithm

Line 13's MoveToFront creates a new order with cluster $\mathcal{C}_y$ at the beginning of the order. Note that running TILO in line 14 is done to get a better estimate of the thick width, rather than to find new pinch clusters. To save computation, lines 13 and 14 can be skipped by using the thick width estimates from the initial TILO run.

The PRC algorithm is presented using the parameter $k$ for the number of partitions. An alternative form of the algorithm is to pass in a threshold value $\epsilon$. Then the while loop would stop when the pinch ratio value $q_i$ in line 7 becomes greater than $\epsilon$ and a variable number of partitions is returned. If a hierarchical clustering is desired then the while loop would run until there are no more split locations. An addition data structure would be used to record the parent-child relationship of the split in line 11.

## 4 Experimental Results

In this section we present a number of experiments to show the efficacy of TILO/PRC[1]. We used one synthetic 2D data set and seven real world data sets. The 2D synthetic example was motivated by [13, Figure 2] with 1200 points sampled from eight distributions. The distributions are described in the appendix and the actual point samples used in the experiments are available at the same location as the PRC source code. The real world datasets include iris, glass, vote, ionosphere, OQ, and UV from [7] and 14cancer from [11]. The OQ and UV data sets are subsets of the overall letter data set. Items with missing attributes were removed and only the training sets were used, if partitioned into training and testing. The attributes were converted into numerical values, but otherwise no feature preprocessing was done. Gaussian similarity function $s(\boldsymbol{x}_i , \boldsymbol{x}_j ) = \exp(\frac{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2})$ was used for the iris, 2Dsyn, and vote datasets. The $k$-nearest neighbor similarity was used for the remaining datasets. The $k$-nearest neighbor similarity used is 1 for both points if either point is one of the $k$ nearest to the other point, and 0 otherwise. The values of $k$ and $\sigma$ were chosen using the heuristics presented in [16] of $k \approx \log(N) + 1$ and $\sigma$ being the average distance to the $k$th nearest neighbor where $N$ is the number of data points.

Clustering algorithms DBScan [5], Affinity Propagation [8], Mean Shift [2], Spectral Clustering [21], and K Means [10] were used for comparison. The scikit [18] implementations of the algorithms were used. The parameters settings for each the algorithms was generated by scikit's implementation. The data sets have class labels which were used to evaluated the performance of the clustering algorithms. The number of classes of each data set was used as the desired number of clusters. The performance was measured by *purity* and *Adjusted Rand Index* (ARI) [12]. Purity is the classification rate with the majority vote of a cluster selecting the cluster's class label. It is easy to understand, but fails as the number of predicted clusters becomes larger than the number of class labels. The ARI is a popular approach to compare the similarity of two different partitions. The predicted partitions were compared to the partition created from the ground truth class labels.

The first set of experiments investigated selecting the partitions from the lexicographic order generated by TILO using the best pinch ratio (3.2) values and using the best normalized cut (3.1) values on the iris data set. The iris data set has three classes with one class linearly separable from the other two. One thousand random initial orderings were processed by TILO and for each, three clusters were formed by selecting the local minimums with the best pinch ratio values and the best normalized cut values. A histogram of the purity values of the resulting partitions are plotted in Figure 4a for pinch ratio and in Figure 4b for normalized cut. The peak for pinch ratio is higher than the peak for normalized cut (0.96 vs. 0.89). This pattern was

[1]The C++ source code for running PRC and TILO is available at `http://www.cs.okstate.edu/~doug/src/prc`. An initial Python binding is also available.

repeated in the other data sets with the exception of the two letters sets OQ and UV. In those cases the peaks were so low that it does not seem relevant (see Figures 4h and 4i for the ARI histograms). The ARI histograms of running TILO/PRC on one thousand random initial orderings of the data sets is presented in Figure 4.

The first experiment demonstrated that the pinch cluster definition used in TILO can correspond to meaningful clusters in real data sets. Since both the pinch ratio and the normalized cut measures used the same sets of possible cut locations from TILO, we conclude that the extra topological information in the linear ordering does provide useful information for creating clusters and that the thick width of the pinch ratio is a reasonable measure of a cluster.

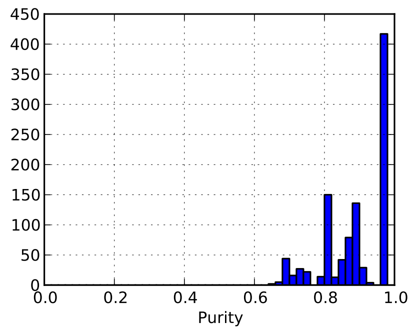
The second set of experiments involved running TILO ten times on each data set with random initial linear orderings, then selecting the final ordering with the smallest lexicographic width. TILO was allowed to completely reduce the lexicographical order (TILO can also stop as soon as the local minima and maxima are determined). This order was then used to determine the clusters using pinch ratio and using normalized cut. The other clustering algorithms were also applied to each data set. Ten separate trails were also used for K means. The number of desired clusters was supplied to TILO/NCUT, TILO/PRC, K Means, and spectral clustering. The other algorithms automatically determined the number of clusters. In the third set of experiments, the cluster labels from the other algorithms were used to set the initial ordering for TILO by grouping points with the same label into a contiguous segments. The results from both experiments are presented in Figure 5. The brown bars present the performance of the original algorithm. The orange bars present the performance of TILO/PRC with the initial ordering based on the clusters from the other algorithm. Performance was measured using ARI. For some of the data sets, the poor performance of DBscan, Affinity Propagation, and/or Mean Shift is due to the mismatch of the number of predicted classes and the number of actual classes. For example on the iris data set, Affine Propagation predicted 21 clusters yielding a purity of 0.97 but an ARI of only 0.19. On iris, both DBscan and Mean Shift predicted two classes. It is assumed that the performance of these algorithms could be improved by tuning some of their parameters instead of using the default settings.

To help understand the effect of using another algorithm's clustering to seed the initial linear order, histograms of running TILO/PRC on one thousand random initial orderings of the data sets is presented in Figure 4. For the iris, vote, and cancer data sets, the results of the initialized TILO/PRC end up in the peak area of the histograms. For those sets, the initialization does not help as TILO/PRC was already performing well from the random initialization. For 2Dsyn and glass, the initialization help select a partitioning at the high edge of the histogram (0.63 for 2Dsyn and 0.27 for glass). For ionosphere, the initialization did not help and for most of the algorithms, they performed worse than the best of ten random initialization. This may be the result of the heavy tail on the lower side of the peak in the histogram. The OQ and UV data sets are interesting in that the original performance of all of the algorithms had very poor ARI. The scores are so close to zero that they don't look like they are plotted on the graphs in Figures 5g and 5h. However, TILO/PRC found a very good partitioning for OQ when initialized with DBScan or Mean Shift and found a very good partitioning for UV when initialized with Affinity Propagation or Mean Shift. The purity (classification accuracy) of the UV clusters is 0.997 for both and of the OQ clusters is 0.925 and 0.89. Those scores would not be bad for a supervised learning algorithm and are outstanding for an unsupervised learning algorithm. As can be seen from Figures 4i and 4h, the histograms for both UV and OQ are very heavy close to zero implying for those data sets randomly selecting an initial order that converges to a good partitioning is unlikely.

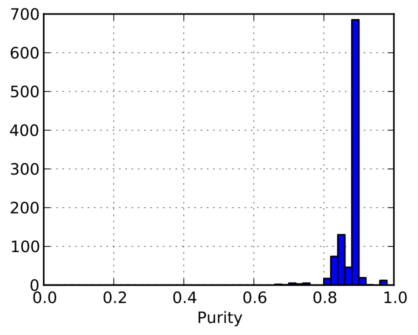
Table 1 presents the average CPU times of 1000 runs of TILO on the data sets. In addition, synthetic EvenOdd graphs were timed. In the EvenOdd graph, each node has 100 random edges of which 70 are to nodes with the same parity and 30 to nodes with opposite parity. The running time of TILO depends on the number of cyclic shifts needed to reduce the order. The computational cost of a shift depends on the number of nodes involved (updating the order) and their edges (updating boundary and slopes). All runs used a single core of a Intel Xeon E5620 CPU running at 2.40 GHz.

## 5 Conclusion

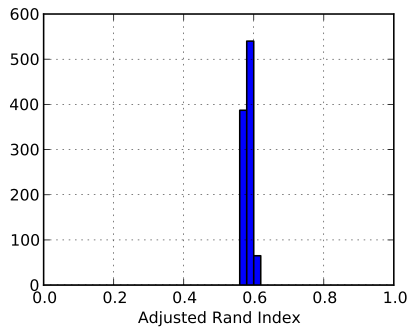
This paper presents TILO and TILO/PRC algorithms. These algorithms provide a novel computational approach for determining clusters based on a principled theoretical foundation in geometric topology. Experiments show that good partitions can be selected from TILO's ordering using either pinch ratio or normalized cut. Pinch ratio has a clear advantage when the size of the partitions are not expected to be equal. Experiments also demonstrate that TILO can converge to a very good partitioning over a number of real world data sets. As a greedy algorithm, TILO can become stuck in a local minimum far from the global minimum. Experiments show that it can be useful to initialized TILO with the clustering results of another algorithm. As the



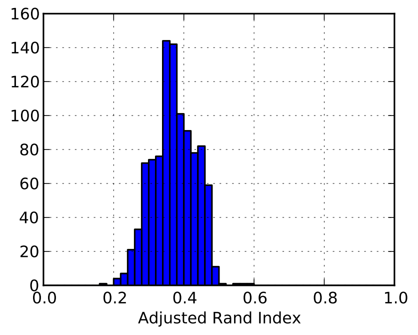
(a) Iris purity using pinch ratio



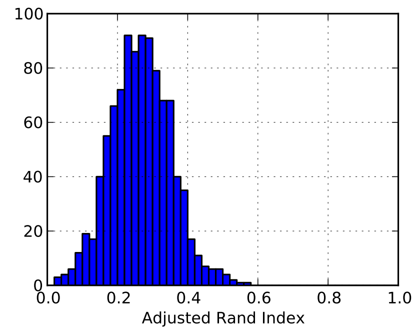
(b) Iris purity using normalized cut



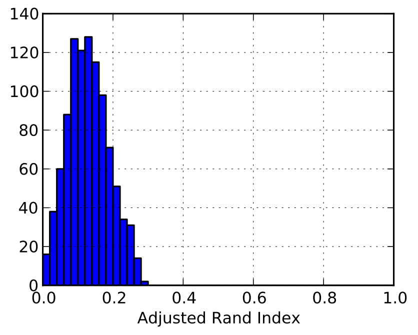
(c) Vote using pinch ratio



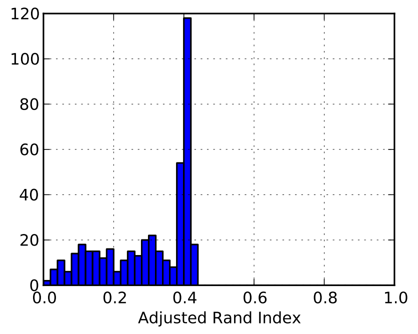
(d) 14cancer using pinch ratio



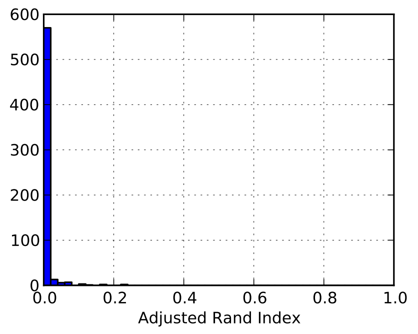
(e) 2Dsyn using pinch ratio



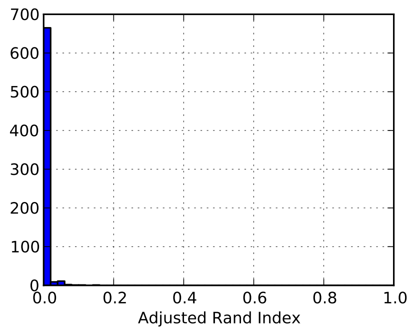
(f) Glass using pinch ratio



(g) Ionosphere using pinch ratio



(h) OQ using pinch ratio



(i) UV ARI using pinch ratio

Figure 4: Histograms of the quality of clusters form from the strongly irreducible orderings generated by TILO on runs over one thousand random initial orderings.

computational cost of PRC and TILO is not large, the authors suggest that data analysts try refining their results with TILO/PRC.

For very large data sets, the linear scan of TILO can become a bottom neck. In future work, we plan to investigate a $k$ branched ordering instead of a linear ordering. This could enable a pairwise branch update that would scale to very large data sets. Another future direction will be to incorporate semi-supervised learning and outlier detection into TILO/PRC.

## References

[1] G. Carlsson and F. Mémoli, *Multiparameter hierarchical clustering methods*, Classification as a tool for research, 2010, pp. 63–70. MR2722123

[2] D. Comaniciu and P. Meer, *Mean shift: A robust approach toward feature space analysis*, IEEE Trans. Pattern Anal. Mach. Intell. **24** (May 2002), no. 5, 603–619.

[3] D. Defays, *An efficient algorithm for a complete link method*, Comput. J. **20** (1977), no. 4, 364–366. MR0478804 (57 #18277)

[4] W. E. Donath and A. J. Hoffman, *Lower bounds for the partitioning of graphs*, IBM J. Res. Develop. **17** (1973), 420–425. MR0329965 (48 #8304)

Table 1: Average CPU Times from 1000 runs used to create the histograms in Figure 4 and 100 runs of synthetic even-odd graphs in which each node has 100 random edges with 70 to same parity nodes and 30 to opposite parity nodes. The $\sigma$ represents standard deviation. The shifts columns contains average total number of cyclic shifts in TILO.

| Data | Nodes | Edges | Time$\pm\sigma$, sec. | shifts |
|---|---|---|---|---|
| 14cancer | 144 | 1134 | 0.002±0.0003 | 2328 |
| glass | 214 | 1788 | 0.006±0.0007 | 5055 |
| ionosph. | 351 | 3496 | 0.025±0.003 | 14000 |
| iris | 150 | 13958 | 0.014±0.002 | 2625 |
| oq | 1536 | 16656 | 0.59±0.04 | 281977 |
| uv | 1577 | 16546 | 0.57±0.04 | 296598 |
| vote | 232 | 53592 | 0.061±0.009 | 4510 |
| 2dmulti | 1165 | 256508 | 3.5±0.34 | 173724 |
| EvenOdd | 250 | 25000 | 0.05±0.005 | 2842 |
| EvenOdd | 500 | 50000 | 0.24±0.03 | 11646 |
| EvenOdd | 750 | 75000 | 0.60±0.09 | 26143 |
| EvenOdd | 1000 | 100000 | 1.2±0.15 | 46646 |
| EvenOdd | 2500 | 250000 | 9.8±1.1 | 290058 |
| EvenOdd | 5000 | 500000 | 49±5.0 | 1156036 |
| EvenOdd | 7500 | 750000 | 145±11.9 | 2591079 |
| EvenOdd | 10000 | 1000000 | 305±28.6 | 4592334 |
| EvenOdd | 12000 | 1200000 | 536±40.1 | 7172175 |
| EvenOdd | 15000 | 1500000 | 710±49.1 | 10306970 |

[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, *A density-based algorithm for discovering clusters in large spatial databases with noise*, Second international conference on knowledge discovery and data mining, 1996, pp. 226–231.

[6] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, *Cluster analysis*, 5th ed., Wiley, 2011.

[7] A. Frank and A. Asuncion, *UCI machine learning repository*, University of California, Irvine, School of Information and Computer Sciences, 2010.

[8] B. J. Frey and D. Dueck, *Clustering by passing messages between data points*, Science **315** (2007), 972–976.

[9] D. Gabai, *Foliations and the topology of* 3*-manifolds. III*, J. Differential Geom. **26** (1987), no. 3, 479–536. MR910018 (89a:57014b)

[10] J. A. Hartigan and M. A. Wong, *A k-means clustering algorithm*, Journal of the Royal Statistical Society, Series C (Applied Statistics) **28** (1979), no. 1, 100–108.

[11] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning*, 2nd ed., Springer Verlag, 2008.

[12] L. Hubert and P. Arabie, *Comparing partitions*, Journal of Classification **2** (1985), no. 1, 193–218.

[13] A. K Jain, *Data clustering: 50 years beyond k-means*, Pattern Recognition Letters **31** (2010), no. 8, 651–666.

[14] J. Johnson, *Topological graph clustering with thin position*, preprint (2012). arXiv:1206.0771.

[15] M. Lackenby, *Heegaard splittings, the virtually Haken conjecture and property* ($\tau$), Invent. Math. **164** (2006), no. 2, 317–359. MR2218779 (2007c:57030)

[16] U. Luxburg, *A tutorial on spectral clustering*, Statistics and Computing **17** (Dec. 2007), no. 4, 395–416.

[17] M. Minoux and E. Pinson, *Lower bounds to the graph partitioning problem through generalized linear programming and network flows*, RAIRO Rech. Opér. **21** (1987), no. 4, 349–364. MR932184 (89e:05159)

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research **12** (2011), 2825–2830.

[19] J. T. Pitts and J. H. Rubinstein, *Existence of minimal surfaces of bounded topological type in three-manifolds*, Miniconference on geometry and partial differential equations (Canberra, 1985), 1986, pp. 163–176. MR857665 (87j:49074)

[20] M. Scharlemann and A. Thompson, *Thin position for* 3*-manifolds*, Geometric topology (Haifa, 1992), 1994, pp. 231–238. MR1282766 (95e:57032)

[21] J. Shi and J. Malik, *Normalized cuts and image segmentation*, IEEE Trans. Pattern Anal. Mach. Intell. **22** (2000), no. 8, 888–905.

[22] R. Sibson, *SLINK: an optimally efficient algorithm for the single-link cluster method*, Comput. J. **16** (1973), 30–34. MR0321382 (47 #9915)

## A Appendix: Experimental Setup

The 2D synthetic data set was generated from eight distributions: a zero mean Gaussian with unit covariance matrix, a Gaussian centered at (0,-6) with a covariance matrix created by scaling the minor axis scale to 1/8 of the major axis and rotated by $\frac{\pi}{6}$. The circles at centered at (6,-5) with radii of 0.5, 1.25, and 2.0 respectively. The spirals are centered at (6,0) with scaling $\frac{1}{2.75\pi}$. The circles and spirals were sampled at uniformly over arc length and corruptive with Gaussian noise with $\sigma = 0.1$ for the circles and $\sigma = 0.02$ for the spirals. The last distribution is uniform noise over the range of the graph. The relative weighting of each distribution is 0.15 for each Gaussian, 0.3 total for all circles, 0.35 total for both spirals, and 0.05 for the uniform background noise.

TILO has the single parameter tiloEpsilon which is a numerical threshold used in comparing floating point values. PRC has the options of policyRefineTILO and policyPRCRecurseTILO corresponding to a flag to completely reduce the width of the order so width can be used to compare multiple runs and to a flag to run lines 13 and 14 for the PRC algorithm to get a better estimate of the thick width. For the experiments, The parameters for all of the TILO/PRC runs used tiloEpsilon=1e-12, policyRefineTILO=1, policyPRCRecurseTILO=0, and seed=3513801. If Gaussian similarity was used for a data set, a threshold of gausssimAdjThreshold=1e-10 was used to zero out small values.

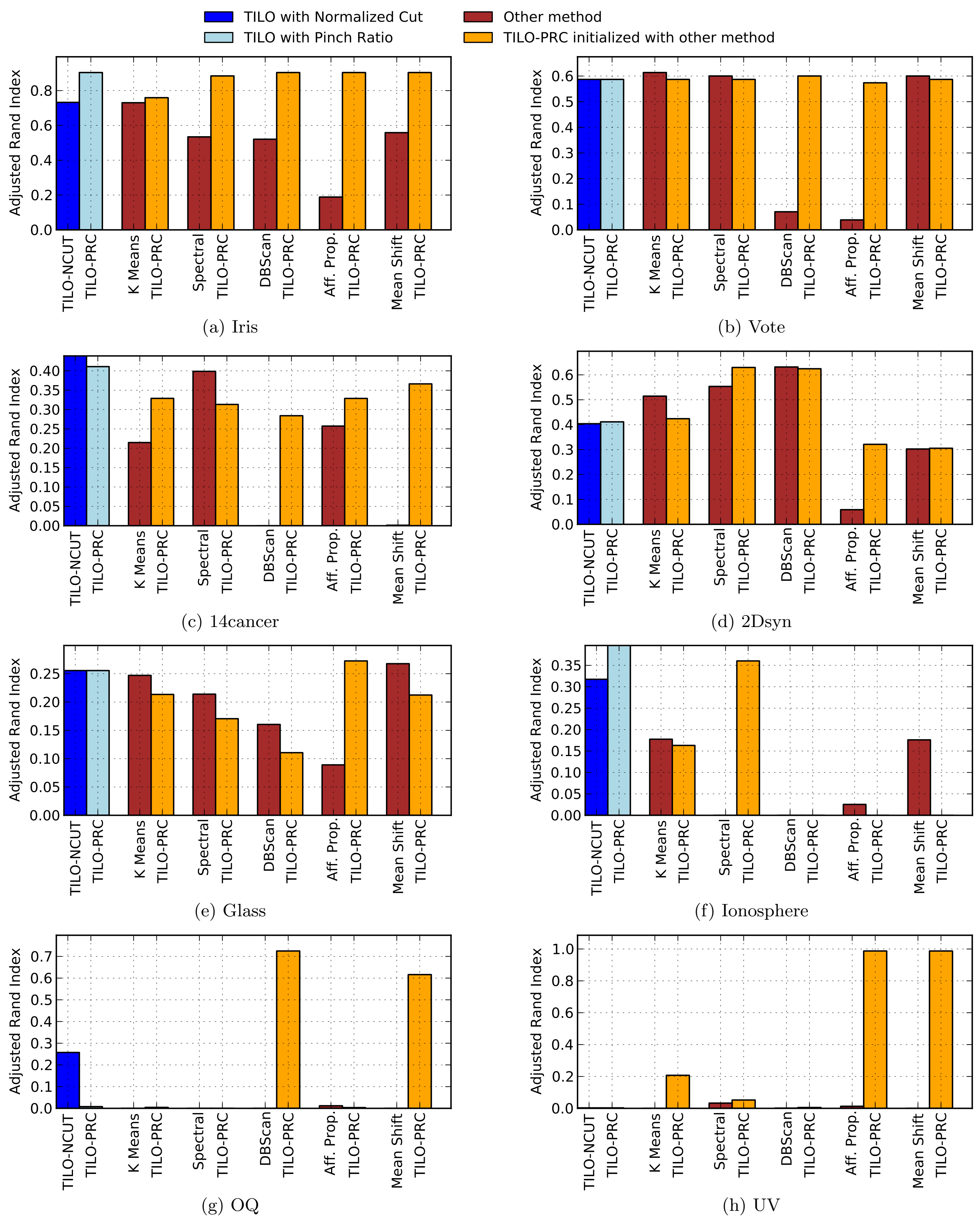


Figure 5: The brown bars present the performance of the each algorithm on each data set. K Means, TILO/NCUT, and TILO/PRC used ten trials. The orange bars present the performance of TILO/PRC when initialized with ordering based on the clustering of the corresponding algorithm.