Chapter 1

Spectral Clustering for Data Streams

1.1 Introduction

Spectral clustering has been empirically shown to perform well under difficult cluster scenarios, but is computationally intensive to run for large data sets. KASP offers an approach to deal with this burden, by summarising the data using k-means, and feeding the centroids as input to spectral clustering. We show how spectral clustering can be applied to online data by combining the Clustream framework with the concepts developed in KASP Yan et al. (2009). Clustream is a framework for dealing with a constant stream of data, and splits the clustering process into two stages: micro-clustering and macro-clustering. We analyse the performance of spectral clustering in an online setting on simulated and real texture-based data sets, both for static and evolving streams. We discuss how to quantify the information loss from summarising the data stream and investigate empirically and algebraically how this perturbation may effect the overall performance.

1.2 Literature Review

Spectral Clustering is a technique for separating data into many groups or clusters. All the elements in each group should be similar, and elements in different groups should be dissimilar. A weighted distance matrix of the data points can be formed, which is called an affinity matrix. By evaluating the spectral properties of this affinity matrix, clusters can be determined. This idea of splitting a data set into groups or clusters, with high within-cluster similarity and between-cluster dissimilarity, can also be thought of as a graph partitioning problem, with the data points as graph vertexes, and the elements in the affinity matrix representing weights on the graph edges. Imagine that all of the data points in the data set are joined together by string. If you were asked break this graph into two distinct pieces, which strings would you cut?

Graph partitioning problems have long been of interest to mathematicians, ever since Euler solved the popular "Seven Bridges of Königsberg" problem in 1736. However the link between graph partitioning and matrix spectra was observed much later when Donath and Hoffman (1973) established a lower bound for the number of edges which must be cut in a graph partitioning problem, based on a factor of the eigenvectors of the affinity matrix. Around the same time Fieldler (1973) published work commenting on the importance of the value of the second eigenvector of the affinity matrix when producing a two way graph partition, also know as a bi-partition. As bi-partitioning was studied the question arose, how to step from bi-partitioning problems to n-partitioning problems? The obvious two approaches are to either recursively apply bi-partitioning on the graph or calculate an n-partition directly from the first n eigenvectors. This is discussed further in Alpert (1995) where they claim that recursive bi-partitioning, which was more popular at the time, is inferior compared with creating one partition using many eigenvectors. For a full review of the historical origins of spectral clustering, see the tutorial von Luxburg et al. (2008).

The first paper to really highlight the use of Spectral Clustering to the Machine Learning community came in 2000. Shi and Malik (2000) treated image segmentation as a graph-cut problem. Mathematically, a cut of a graph G is the set of edges S, such that the removal of S from G disconnects G. A graph is connected if there is a path from every vertex to every other vertex in G. The weight of the cut can be calculated by summing the weights of the edges which will be broken when a cut is made, as given in equation (1.2.1).

$$\text{cut(A,B)} = \sum_{p \in A, q \in B} w(p,q) \tag{1.2.1}$$

The Minimum cut proposed in (Wu and Leahy, 1993) is simply the cut which generates the minimum cut weight. However the minimum cut does not always provide the best solution; it tends to favour removing small sections and isolated nodes of the graph. The main proposal from Malik was to consider the cut function as a fraction of the total edge connections to all nodes in the graph, instead of using the total edge weight connecting the two partitions. This subtle yet important distinction provides us with the highly acclaimed normalised cut (NCut), defined in equation (1.2.3), which is an integral part of spectral clustering algorithms. The main idea is that big clusters will increase assoc(A,V), thus decreasing Ncut(A,B). Minimising the normalised cut is an NP-complete problem but an approximate solution can be found efficiently by finding the eigenvector with the second smallest eigenvalue.

$$\begin{aligned} \text{Ncut(A,B)} &= \tfrac{\text{cut(A,B)}}{\text{assoc(A,V)}} + \tfrac{\text{cut(A,B)}}{\text{assoc(B,V)}} \\ \text{assoc(A,V)} &= \textstyle\sum_{u \in A, t \in V} w(u,t) \end{aligned} \tag{1.2.2}$$

On Spectral Clustering Ng et al. (2001) was one of the first papers to provide theoretical guarantees on performance for spectral clustering algorithms. Unlike previous authors who had only shown empirical results to justify spectral clustering abilities, they prove that their spectral clustering algorithm will produce a reasonable clustering, given certain assumptions that the clusters are well-spaced.

Although spectral clustering has been shown to perform well empirically on simple data sets, computational problems arise as the data set size increases. There have been a number of ways to deal with speed up in the static case, one of the most popular methods is to use Nyström efficiency methods Williams and Seeger (2001) Fowlkes et al. (2004). The Nyström method samples the columns of the affinity matrix and approximates the full matrix by using correlations between the sampled columns and the remaining columns.

An alternative to Nyström methods is to perform some data permutation on the data as a pre-processing step. By feeding a smaller set of representative points into the spectral clustering algorithm instead of the whole data set, we can lessen the effect of the computations bottleneck that comes with eigen analysis. Fast Approximate Spectral Clustering Yan et al. (2009), Huang et al. (2008) explores the theoretical guarantees on misclustering the data set, given that some permutation has been performed on the data before the spectral clustering. Specifically they exactly quantify the misclustering of data sets, given that the original data is summarised using K-means (KASP) or a Random Project tree (RASP) as preprocessors.

Another recent improvement to deal with the computational issues in spectral clustering is proposed in Local Information-based fast approximate spectral clustering Cao et al. (2014). Li-ASP consists of two upgrades a sparse affinity graph to speed up computation and local interpolation to improve clustering performance. The sparse affinity graph effectively uses k nearest neighbour or an ϵ neighbourhood to set many elements in the affinity graph to 0. Local interpolation is suggested based on an issue identified about KASP Yan et al. (2009) that if a representative point is miss-assigned to the wrong cluster, then all members of that cluster will be miss-assigned. Local interpolation can avoid these situations by using weighted differences from the nearest p centres that a data point belongs to. KASP only uses it's assigned cluster.

The choice of scaling parameter σ is discussed in Zelnik-manor et al. (2004). They argue that for data which has a cluttered background, or multi-scale data, one global parameter choice for σ is not sufficient. Before this, the most common method for choice of parameter was manually or Ng et al. (2001) automatically chose σ by running their clustering algorithm repeatedly for a number of values and selecting the one which provides least distorted clusters of the rows of Y. In Zelnik-manor et al. (2004) a localised parameter is calculated. Each data point has it's own σ value calculated based on it's neighbourhood (here $K = 7$). They also present a method for automatically choosing the true number of clusters C using the eigenvectors to inform a choice. More commonly the eigenvalues are commonly used to estimate K but if the groups are not clearly separated, identifying the number of groups from eigenvalues alone is not trivial.

Ning et al. (2007) Ning et al. (2010) Updates eigenvalues incrementally. A change in vertices or edge can be used to algorithmic-ally update the eigenvalues and eigenvectors directly, however the authors recommend a full re-clustering in batch to minimise cumulative errors. The indirect updating of eigenvectors means that the orthogonality property may be lost. Also if the spatial neighbourhoods of often changing vertices are large it can still be computationally hard as the eigenvalue step involves the inversion of a matrix.

1.3 Spectral Clustering

In this Section we provide a brief introduction to Spectral Clustering, and discuss the choice of affinity matrices. The KASP Yan et al. (2009) algorithm will be introduced, which we will use in an online fashion in Section 1.4.

The goal of spectral clustering is to partition the data into k disjoint classes such that each x_i belongs to one and only one class. Data which is compact may be simple to cluster as the gaps between clusters are easier for simple clustering algorithms to identify. Sometimes data may be connected and therefore should hopefully be classed as a whole cluster, but it may not be tightly compact and spherical. Spectral clustering can provide good quality segmentation on even these difficult cases, however its performance comes at the cost of computational complexity. We shall use the Jordan-Weiss (NJW) spectral clustering framework Ng et al. (2001) which is described in Algorithm 1.

We want to use a matrix to represent the pairwise similarities or distances between all data points i and j, called $a_{i,j}$. We refer to the matrix $A = (a_{ij})_{i,j=1}^{n}$ as the affinity matrix. We can think of the affinity matrix A as a metric that determines how close two points are in our space. A popular choice is to define A to be the Gaussian kernel, as defined in equation 1.3.1, where the parameter σ controls the width of the local neighbourhoods which we want to model.

$$a_{i,j} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), i, j = 1, \ldots, n. \tag{1.3.1}$$

If x_i and x_j are very close, then $a_{i,j} \to 1$, and if they are far apart $a_{i,j} \to 0$. This matrix will have ones along the diagonal, and will be symmetric ($a_{ij} = a_{ji}$).

If we mainly wish to model the local relationships, using all of the possible pairwise data connections may not be necessary. It is possible to use a weighted k-nearest neighbour structure to build the affinity matrix (once corrections have been made to ensure that this matrix is symmetrical). Another option is to choose some threshold ϵ and only consider connections between data points whose pairwise distances are smaller than ϵ. This is an ϵ-neighbourhood graph. Although we can weight by ϵ, if we choose ϵ to generate a small ϵ-neighbourhood, then the differences between the weights will be small/negligible. Therefore a simple construction of the ϵ-neighbourhood graph is as follows.

$$a_{ij} = \begin{cases} 1, & \text{if } d_{ij} < \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{1.3.2}$$

Using this construction would give a sparse affinity matrix instead of a fully connected graph, which will help lower the computational complexity.

The degree of each vertex v_i is defined as, $d_i = \sum_{j=1}^{N} A_{ij}$, the sum of the rows of the affinity matrix. The degree matrix, D, is then defined as a diagonal matrix with i^{th} diagonal element equal to d_i. We use the Normalised Laplacian defined in equation 1.3.3, which relates to an approximation of minimising the Normalised Cut, discussed in Section 1.2. Ideally, the Laplacian L will be block-diagonal, where each block refers to a particular cluster.

$$L = D^{-1/2} A D^{-1/2} \tag{1.3.3}$$

Spectral clustering can be challenging for very large data sets, constructing the affinity matrix A and computing the eigenvectors of L have computational complexity $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ respectively. Yan et al. (2009) proposes a fast approximate spectral clustering which uses a k-means pre processing step to lessen the computational complexity whilst retaining good clustering performance. Firstly k-means is run on the whole data set where k is chosen to be large but such that $k \ll n$. The centres of the clusters are then used as representative data points for the whole data set. Spectral clustering is performed on the representative set only, which is faster than performing spectral clustering on the full data set. The resulting

```
Algorithm 1 NJW spectral clustering algorithm
Input: Data set $S = x_1, \dots, x_n$, number of clusters $k$
Output: $k$-way partition of the input data
1: Construct the affinity matrix $A$ by the following Gaussian kernel function:
       $$a_{i,j} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), i, j = 1, \dots, n.$$
2: Compute the normalised affinity matrix $L = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, where $D$ is the diagonal matrix
   with $D_{ii} = \sum_{j=1}^{n} a_{ij}$.
3: Compute the $k$ eigenvectors of $L$, $v_1, v_2, \dots, v_k$, associated with the $k$ largest eigenvalues,
   and form the matrix $X = [v_1, v_2, \dots, v_k]$.
4: Renormalize each row of $X$ to form a new matrix $Y$.
5: Partition the $n$ rows into $k$ clusters via a general cluster algorithm, such as the k-means
   algorithm.
6: Assign the original point $x_i$ to the cluster $k \iff$ the corresponding row $i$ of the matrix
   $Y$ is assigned to the cluster $k$.
```

cluster labels for the representative data are linked back to the original data set such that every original data point acquires the same label as its associated k-means cluster center. The KASP algorithm is repeated in Algorithm 2 with modified notation.

1.4 Micro-cluster based spectral clustering in data streams

A relatively new challenge to clustering is working with data streams. Data stream clustering is clustering of data which arrives in an ordered sequence, continuously; for example, sensor data or shopping transactions. Much work has been done developing offline clustering methods, such as spectral clustering, but it is not suitable to apply these offline methods to the streaming scenario. Simply running an offline clustering algorithm on all the data observed so far may not be feasible. Often, the data cannot be stored in its entirety, therefore the older data has to be thrown away to make room for the new arrivals. This might at first seem like not a big issue as this will naturally weight the data temporally. However we may still wish to use older data in current clustering. Also clustering algorithms can be computationally expensive. For example, spectral clustering has $\mathcal{O}(n^3)$ when solving the eigenvalue problem. Therefore we need to be able to update our idea of the data as new points arrive efficiently and simply with little computational issues. Also we have the issue that we will only see

Algorithm 2 KASP

```
Input: $n$ data points $\{xi\}_{i=1}^{n}$, number of representative points $k$
Output: $m$-way partition of the input data
1: Perform k-means with k clusters on $x_1, \dots, x_n$ to:
2: a) Compute the cluster centroids $y1, \dots, y_k$ as the $k$ representative points.
3: b) Build a correspondence table to associate each $x_i$ with the nearest cluster centroids $y_j$
   .
4: Run a spectral clustering algorithm on $y_1, \dots, y_k$ to obtain an $m$-way cluster membership
   for each of $y_i$.
5: Recover the cluster membership for each $x_i$ by looking up the cluster membership of the
   corresponding centroid $y_j$ in the correspondence table.
```

the data once (one-pass). This can be imagined as data flying past in a chute. You can look at the data as it flies by, but once it has passed by, it cannot be accessed again. Some algorithms such DBScan require many passes or iterations of the data, therefore these type of clustering methods are not directly suitable for the online data streaming case.

Clustream Aggarwal et al. (2003) offers a framework which allows quick and easy updates and the ability to perform sophisticated clustering algorithms. The main idea is to separate the clustering process in two stages; a micro clustering which continuously updates statistical summaries of the data set, and a macro clusterer which is run on a user request. The micro-clusters are defined as a temporal extension of the cluster feature vector Zhang et al. (1996).

Assume that we have a data stream S which consists of d-dimensional data x_i arriving in sequence. $S = \{x_1, x_2, x_3, \dots x_i, \dots, \}, x_i \in \mathbb{R}^d$

Keeping track of all of the data is very hard. As we have mentioned above, we only have one-pass access to the data. This is where the idea of micro clusters comes in. Micro clusters are a way of maintaining an active, evolving representative summary of your data, without actually having to store the absolute values for all the data points. This data summary can then be used on a user request to perform a macro clustering using the summarised data rather than the full data set. If the micro clusters represent the true underlying data stream well, then the difference between the clustering on the summarised data and the true full data should be small. This was shown in Yan et al. (2009).

How does this micro-clustering work? The data stream is summarised by many small clusters, which are initially generated by k-means. We take an initial training set, and perform k-means but choose the value of k to be much larger than the expected number of true macro clusters. The aim here is to create a fine scale summary of the data. The value of k should be chosen to be as large as computationally comfortable. The larger k is, the finer scale that the summaries will be. It is vital to ensure that the micro-cluster well represent the underlying data set or else the macro clustering will under perform.

These k clusters are our first micro clusters. Over time, we will update these micro clusters. Adding new data points to them, and sometimes removing old micro clusters to make the way for new micro clusters (the number of micro clusters should stay fixed throughout).

Each micro-cluster j is stored as a $(2 \cdot d + 3)$ tuple $(CF1^x, CF2^x, n, CF1^t, CF2^t)$. $CF1^x$ is the sum of all observed data in that micro cluster, $CF2^x$ is the sum of the squares of the data and n is the number of elements in the micro-cluster. $CF1^t$ and $CF2^t$ refer to the sum of the time stamp, and the sum f squared time stamps respectively. Note that both $CF1^x$ and $CF2^x$ will have d entries, one for each dimension.

Micro-cluster j will have

$$CF1^x[p] = \sum_{x_i \in j} x_i^p,$$
$$CF2^x[p] = \sum_{x_i \in j} (x_i^p)^2.$$

These updates only require addition therefore they are cost effective. Critically it is possible to use these summaries to calculate the centre of each micro-cluster. It is these centres which as used as representative points for input into the macro clustering. As new points in the data stream arrive, they are either allocated to a micro-cluster and the update procedure discussed above it carried out, or a new micro-cluster is created. The decision for a new micro cluster to be created is based on whether the new data point is close enough to it's nearest cluster centre.

When a new data point arrives, it's nearest micro-cluster is identified using the Euclidean distance metric. If the data point falls within the Maximum Boundary Factor of it's nearest cluster centre, then it is absorbed as part of that cluster. If not, it is used to create a new micro-cluster. However we stated earlier than the number of micro-clusters must remain fixed throughout the process. Therefore if a new micro-cluster is formed, either an existing micro-cluster must be deleted, or two close micro-clusters should be merged. We follow the methodology in Clustream by first looking for an “old” micro-cluster to delete (using the time-stamp references detailed in the original paper) and otherwise combine the two nearest micro-clusters. In this way, the algorithm should be able to track the data stream as it evolves.

With this online micro-cluster maintenance, the data stream should remain well represented over time. When a new data point arrives, if it is the start of a new evolving cluster, it will be allowed to grow, however if it is an outlier, no more points will be added to it, and over time it may be deleted from the system all together.

Clustream has proved popular over time. Since the paper was first published in 2003, it has been cited over 1400 times. Empirically there is much evidence to show that micro-clustering is an effective way to maintain information about your data set. However there has been little theoretical work to explain why this micro-clustering method seems to work so well.

Our intention is to a develop method for applying spectral clustering to a data stream. In order to achieve this we adapt a micro-clustering type approach to quickly update a summary of the data. When an overall clustering is required, spectral clustering is performed using the centres of the micro-clusters as the input data. The micro-clusters act as a way of summarising the constantly arriving data steam whilst allowing updates to occur in a non intrusive, non-computationally difficult manner, with limited storage requirements.

Our algorithm is as follows:

Algorithm 3

```
  Initialise micro-clusters
  New data point arrives, update micro-clusters in the manner discussed. Either allocate
  the data point to an existing micro-cluster or initialise a new micro-cluster.
  At certain intervals take the centres of the micro-clusters as input to a spectral clustering
  algorithm.
  Then link the spectral clustering output back to the original data points (if required) either
  simply by assuming that each cluster member takes the same result as its cluster centre,
  or a more complicated weighted method.
```

How should we use the centres as input into spectral clustering? Silva et al. (2013) suggests two ways of using the centres as input; use them directly, or scale them by the number of elements in a micro cluster. The nature of this algorithm allows the user to get close to online streaming and perform spectral clustering on a summary of the whole of the data set.

1.5 EM as a micro clustering technique

1.6 Experimentation

1.7 Conclusion

Bibliography

Charu C. Aggarwal, T. J. Watson, Resch Ctr, Jiawei Han, Jianyong Wang, and Philip S. Yu. A Framework for Clustering Evolving Data Streams. *Proceedings of the 29th international conference on Very large data bases*, pages 81–92, September 2003. doi: 10.1.1.13.8650. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.8650`.

So-Zen Yao Charles J. Alpert. Spectral Partitioning: The More Eigenvectors, The Better. In *32nd Design Automation Conference*, pages 195–200. ACM, December 1995. ISBN 0-89791-725-1. doi: 10.1109/DAC.1995.250089. URL `http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1586701`.

Jiangzhong Cao, Pei Chen, Qingyun Dai, and Wing Kuen Ling. Local information-based fast approximate spectral clustering. *Pattern Recognition Letters*, 38(1): 63–69, March 2014. ISSN 01678655. doi: 10.1016/j.patrec.2013.11.005. URL `http://www.sciencedirect.com/science/article/pii/S0167865513004285`.

W. E. Donath and A. J. Hoffman. Lower Bounds for the Partitioning of Graphs. *IBM Journal of Research and Development*, 17(5):420–425, September 1973. ISSN 0018-8646. doi: 10.1147/rd.175.0420. URL `http://dl.acm.org/citation.cfm?id=1664638.1664644`.

Miroslav Fieldler. Algebraic Connectivity of Graphs. *Czechoslovak Mathematical Jornal*, 23(2):298–305, 1973. ISSN 0011-4642. doi: 10.1080/03081080500054810. URL `https://eudml.org/doc/12723`.

Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the Nyström method. *IEEE transactions on pattern analysis and machine intelligence*, 26 (2):214–25, February 2004. ISSN 0162-8828. doi: 10.1109/TPAMI.2004.1262185. URL `http://www.ncbi.nlm.nih.gov/pubmed/15376896`.

Ling Huang, Donghui Yan, Nina Taft, and Michael I Jordan. Spectral clustering with perturbed data. In *Advances in Neural Information Pro-*

cessing Systems, pages 705–712, 2008. ISBN 9781605609492. URL `http://papers.nips.cc/paper/3480-spectral-clustering-with-perturbed-data`.

Andrew Y Ng, Michael I Jordan, and Yair Weiss. On Spectral Clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, pages 849–856, 2001. doi: 10.1.1.19.8100. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.8100`.

Huazhong Ning, W Xu, Y Chi, Y Gong, and Ts Huang. Incremental Spectral Clustering With Application to Monitoring of Evolving Blog Communities. *SIAM International Conference on Data Mining*, pages 261–272, 2007. URL `http://www.appliedmathematician.org/proceedings/datamining/2007/dm07_024Ning.pdf`.

Huazhong Ning, Wei Xu, Yun Chi, Yihong Gong, and Thomas S. Huang. Incremental spectral clustering by efficiently updating the eigen-system. *Pattern Recognition*, 43 (1):113–127, January 2010. ISSN 00313203. doi: 10.1016/j.patcog.2009.06.001. URL `http://dl.acm.org/citation.cfm?id=1595888.1595943`.

Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. ISSN 01628828. doi: 10.1109/34.868688. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=868688`.

Jonathan a Silva, Elaine R Faria, Rodrigo C Barros, Eduardo R Hruschka, André C P L F De Carvalho, and João Gama. Data stream clustering. *ACM Computing Surveys*, 46(1):1–31, 2013. ISSN 03600300. doi: 10.1145/2522968.2522981. URL `http://doi.acm.org/10.1145/2522968.2522981\nhttp://dl.acm.org/citation.cfm?doid=252296`

Ulrike von Luxburg, Mikhail Belkin, and Olivier Bousquet. Consistency of spectral clustering. *The Annals of Statistics*, 36(2):555–586, April 2008. ISSN 2168-8966. URL `http://projecteuclid.org/euclid.aos/1205420511`.

Christopher K.i. Williams and Matthias Seeger. Using the Nystrom Method to Speed up Kernel Machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688, 2001. URL `http://papers.nips.cc/paper/1866-using-the-nystrom-method-to-speed-up-kernel-machines`.

Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Ma-*

chine Intelligence, 15(11):1101–1113, 1993. ISSN 01628828. doi: 10.1109/34.244673. URL `http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=244673`.

Donghui Yan, Ling Huang, and Michael I Jordan. Fast approximate spectral clustering. In *Work*, page 907, New York, New York, USA, June 2009. ACM Press. ISBN 9781605584959. doi: 10.1145/1557019.1557118. URL `http://portal.acm.org/citation.cfm?doid=1557019.1557118`.

Lihi Zelnik-manor, Lihi Zelnik-manor, Pietro Perona, and Pietro Perona. Self-tuning spectral clustering. *Advances in Neural Information Processing Systems 17*, pages 1601–1608, 2004. doi: 10.1.1.84.7940. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.84.7940`.

Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *ACM SIGMOD Record*, 25(2): 103–114, June 1996. ISSN 01635808. doi: 10.1145/235968.233324. URL `http://dl.acm.org/citation.cfm?id=235968.233324`.