GRAPHS, CLUSTERING AND APPLICATIONS

DERRY TANTI WIJAYA

(B. Comp. (Hons), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE

2008

ACKNOWLEDGMENTS

I would like to thank my supervisor for giving me the gift of research, for the sheer joy of it and the freedom of mind it brings. No amount of words (or postcards) can ever express my gratitude to you. Thank you for believing in me and making *me* believe in me.

Thank you to Komo/Bebek/Aal for always being the source of my inspiration. Without your constant patronizing and creative ideas this thesis may have been completed years later. Thank you to my parents, grandmas and grandpa for letting me live and for stories told when I was young, and to my monster siblings - Ghe, Krun, Nyit and Nyo – without whose distraction this thesis would have been completed weeks earlier.

Thank you to all the Komodo dragons that have been living in the Komodo Island for the past I-am-not-quite-sure-how-many years, but thank you anyways for always being there *not* for me.

Lastly, thank you to my professors and my friends for your valuable advice. I am taking a *detour* now but I will be back.

"The game will never be over, because we're keeping the dream alive" -- Freiheit

TABLE OF CONTENTS

SUMMARY

Graphs are important data structures in areas such as data mining, natural language processing, information retrieval, and information extraction. Graphs can be used to model pairwise relations between objects. Graphs contain vertices that represent objects and edges connecting the vertices that represent pairwise relations between the objects.

Structures that can be modeled by graphs are ubiquitous. For example, in clustering, similarities between objects to be clustered can be expressed as a weighted graph where objects are vertices and similarities are the weights of edges. Clustering then reduces to the problem of graph clustering. A natural notion of graph clustering is the separation of sparsely connected dense sub graphs from each other based on the notion of intra-cluster density vs. inter-cluster sparsity.

Since many problems of practical interests, such as clustering, can be modeled by graphs, the applications of graph algorithms are numerous. The purpose of our research is to study and gain insights into graph algorithms, graph-based clustering and their applications. We study graph algorithms that we believe to be related to clustering problem, namely the random walk and minimum spanning tree algorithm. We study existing graph-based clustering algorithms: Star clustering, Markov clustering, and minimum spanning tree clustering and their applications. We also explore independently other applications of random walk algorithm in opinion mining and time series correlation to illustrate the wide-range applicability of the random walk algorithm.

In graph algorithms, we propose a novel variant of randomized minimum spanning tree algorithms. In graph-based clustering, we propose a family of Star clustering variants, and two novel families of graph-based clustering algorithms: MST-Sim and Ricochet. Our variants of Star clustering explore various techniques, including random walk algorithm, to identify centroids (Star centers). MST-Sim explores minimum spanning tree algorithm for clustering, adding to it intra- and inter-cluster similarity metrics that have basis in graph theory. Ricochet uses results of our study on Star clustering to identify centroids and our study of minimum spanning tree

algorithm to identify edges to merge clusters. Ricochet also uses the idea of vertices reassignment from K-means to reach terminating condition. Combining these ideas results in Ricochet being unconstrained – it does not require any parameter to be defined a priori – and able to maximize intra-cluster similarities. In applications, we have illustrated and evaluated the performance of our graph-based clustering algorithms for the task of document clustering, k-member clustering, and clustering for part-of-speech tagging. We have also proposed a novel method based on random walk to rank items according to opinions in their reviews and a novel method that combines techniques from machine translation and random walk to measure time series correlation.

We perform comprehensive empirical evaluation on our proposed methods and provide theoretical analysis on their performance. We show that our proposed methods are effective and efficient in their domains of applications.

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

Section 1.1 Graphs

Graphs are mathematical structures used to represent pairwise relations between objects from a certain collection. Graphs contain vertices (that represent objects) and edges connecting the vertices (that represent pairwise relations between the objects). A graph G is therefore an ordered pair G = (V, E) where V is a set of vertices and E is a multiset of edges: unordered pairs of vertices. A simple, weighted graph is an undirected graph that has no self-loops and no more than one edge between any two vertices: i.e. edges form a set rather than a multiset and each edge is an unordered pair of *distinct* vertices. A number or weight is assigned to each edge, representing the weight of relation between the two vertices connected by the edge.

Clustering is the unsupervised process of discovering natural clusters so that objects within the same cluster are similar and objects from different clusters are dissimilar according to some similarity measurements. In a vector space, for instance, where objects are represented by feature vectors, similarity is generally defined as the cosine of the angle between vectors. In clustering, if similarity relations between objects are represented as a simple, weighted graph where objects are vertices and similarities between objects are the weights of edges; clustering reduces to the problem of graph clustering. A natural notion of graph clustering is the separation of sparsely connected dense sub graphs from each other based on the notion of intra-cluster density vs. inter-cluster sparsity. Clustering is the separation of dense from sparse regions of space or components of graphs. Clusters are dense regions or components.

Since many problems of practical interests, such as clustering, can be represented by graphs, the application of graph algorithms is numerous. Specifically for clustering, we identify two important graph algorithms that may be related to the problem of clustering, namely random walk and minimum spanning tree algorithms.

Intuitively, random walk algorithm can produce ranking of vertices in the graph thus identify potentially significant vertices. Minimum spanning tree algorithm can identify potentially significant edges or path in the graph. These are issues closely related to the problem of clustering, namely the identification of significant vertices as candidate centroids (seeds) and the identification of significant edges to merge or split clusters.

Our contribution is the study of graph algorithms and their relationships to clustering. In addition, we look independently at other applications of random walk in the domain of opinion mining and time series correlation, to illustrate the wide-range applicability of random walk algorithm on graphs.

Section 1.2 Applications

Since many problems of practical interests can be represented by graphs, the application of graph algorithms is numerous. For clustering, weighted graphs in which pairwise relations have some numerical values can be used to represent the clustering problem. Vertices are objects to be clustered and each edge is weighted by some similarity measures of the two vertices connected by the edge. We call clustering algorithms that are based on such graph representation, graph-based clustering algorithm.

The best known graph-based clustering algorithm is Zahn's Minimum Spanning Tree (MST) clustering [1]. The algorithm is based on constructing the MST of the graph and then deleting the MST edges with the largest length to generate clusters [2]. The hierarchical clustering algorithms [3] are also related to graph-based clustering [1]. Another well known graph-based clustering algorithm is Markov clustering [4] that simulates stochastic flow or random walks in the graph. The flow is eventually separated into different regions, yielding a cluster interpretation of the initial graph.

Aside from the traditional document clustering problem [5, 6, 7, 8], clustering at sentence level has been applied in the diversity-based unsupervised text summarization [9]. In the World Wide Web (WWW), clustering can be applied to document classification, automatic generation of taxonomy of web documents, and clustering web log data to discover similar access patterns [10]. Clustering has also been applied in areas such as natural language processing and privacy-protection (k-anonymity).

Random walk algorithm on graphs has also wide-range applications. For example, in the WWW, the link structure of the WWW can be represented by a directed graph, where vertices are the web pages and a directed edge from page A to page B exists if and only if page A contains a link to page B. Graph representation of this link structure has given rise to most notably, Google's PageRank algorithm [11], which applies random walk to the graph in order to rank the vertices and therefore ranking the web pages. Random walk on graphs has also been used to compute marginalized kernel (or similarity) between labeled graphs [12].

Section 1.3 Contributions

The purpose of our research is to study graph algorithms, graph-based clustering algorithms, and their applications.

From our study of graph algorithms, we identify to important graph algorithms – namely random walk and minimum spanning tree algorithm – which we believe to be related to the issues of clustering. Random walk has been used to identify significant vertices in the graph that receive maximum flow [11] while minimum spanning tree algorithm has been used to identify significant edges in the graph [2]. We believe these two graph algorithms have useful applications in clustering, namely for identifying centroids (and correspondingly the number of clusters) and for identifying edges to merge or split clusters such that intra-cluster similarity is maximized while inter-cluster similarity is minimized.

In some graph-based clustering algorithms, the parameter for determining the number of clusters is defined a priori and explicitly [2, 3] while in others it is defined implicitly as a fine-tuning parameter [4]. In some other graph-based clustering algorithms, although the parameter for determining the number of clusters is not needed, the parameter for choosing edges to merge or split clusters needs to be defined a priori [13].

Based on our study of graph algorithms and graph-based clustering algorithms, we propose novel variants of Star clustering algorithm that use different techniques for identifying centroids (Star centers), and two novel graph-based clustering algorithms: MST-Sim and Ricochet.

We use random walk as one technique to identify potential centroids in our variants of Star clustering. Our novel variants of Star clustering algorithm improve significantly the effectiveness of the original Star clustering algorithm without affecting efficiency for the problems of document clustering.

We use minimum spanning tree algorithm for our MST-Sim clustering and adds to it intra-cluster and inter-cluster similarity metrics that have basis on graph theory. Our variants of MST clustering achieve higher performance in terms of effectiveness and efficiency than other state-of-the-art graph-based clustering algorithms for the problems of document clustering.

We propose a novel clustering algorithm, Ricochet, which does not require any parameter to be defined a priori. Instead, Ricochet uses results of our study on Star clustering to choose centroids automatically and our study on minimum spanning tree to choose edges automatically. Ricochet also uses ideas from K-means to determine its termination condition. While the fact that Ricochet is unconstrained already has an advantage of its own, Ricochet also performs respectably well in terms of effectiveness and efficiency for the problems of document clustering compared to other state of the art graph-based clustering algorithms.

We explore the applications of our proposed clustering algorithms for the task of document clustering, privacy-protection (using k-member clustering), and part-of-speech tagging (using clustering). In future other evaluation metrics in other domains can be considered, for example in clustering for the domain of biomedical or the web

Based on our study of graph algorithms, we also look into the applications of random walk algorithm in opinion mining and in computing time series correlation. We also propose a variant of randomized minimum spanning tree algorithm.

Section 1.4 Plan of Thesis

In chapter 2 we present related works on graph algorithms that we study, namely random walk and minimum spanning tree algorithm. We also present related works on state of the art clustering algorithms, including graph-based clustering algorithms. We further present related works on the application domains that we are exploring: document clustering, k-member clustering for privacy-protection, clustering for part-of-speech tagging, opinion mining and time series correlation.

In chapter 3 we present our proposed methods on graph algorithms: a variant of randomized minimum spanning tree algorithm, graph-based clustering algorithms, and applications on document clustering, k-member clustering, clustering for part-of-speech tagging, opinion mining and time series correlation.

In chapter 4, we present our experimental setup – how we illustrate and evaluate the performance of our proposed methods. In future, scalability evaluation and indirect method of evaluation can also be considered. For example, an indirect method of evaluation by evaluating whether clustering can be used in the re-ranking of results to improve ranking of retrieval results.

In chapter 5, we present the results of our experiments and evaluation of the proposed methods. We conclude in chapter 6.

CHAPTER 2. RELATED WORKS

Section 2.1 Graphs

2.1.1 Graph Algorithms

In this section we review graph algorithms that we study: random walk and minimum spanning tree algorithm.

2.1.1.1 Random Walk

Suppose we are given a connected, weighted graph G = (V, E). Suppose that the graph is bi-directed, i.e. $(x, y) \in E$ if and only if $(y, x) \in E$. Suppose also that the graph has no loops, so that $(x, x) \notin E$ for any $x \in V$. Let $N(x) = \{y \in V: (x, y) \in E\}$ be the set of neighbors of a vertex x. Since G is a weighted graph, there is a weight $c(x, y) > 0$ associated with each edge $(x, y) \in E$. The weight is symmetric such that $c(x, y) = c(y, x)$ for $(x, y) \in E$. Extend the weight c to a function on all V x V by defining $c(x, y) = 0$ for $(x, y) \notin E$. If, as an analogy, we imagine a fluid flows through the edges of the graph; the weight measures the capacity of the edge.

Let $C(x) = \Sigma_{y \in V} c(x, y)$ and suppose that $C(x) < \infty$ for each $x \in V$. The Markov chain with state space V and transition probability matrix P given by

$$P(x, y) = c(x, y) / C(x), \quad (x, y) \in V^2 \qquad (1)$$

is called a random walk on the graph G [14]. This chain governs the movement of a random walker moving along the vertices of G. If the walker is at vertex x at a given time, the walker will be at a neighbour of x at the next time; the neighbor is chosen randomly, in proportion to the weight of the edges connecting x to its neighbors.

Random walk has been applied to computer science, economics and various other fields to represent random processes in time. Random walks on graphs have been applied to rank the

vertices in the graph [11], to compute expected similarity between labeled graphs [12], or to identify clusters in a graph [4].

Random walk on graphs has been used to rank vertices in a graph. Given a graph where edges are weighted by the probability of traversing from one vertex to another, the rank of the vertices can be computed according to the structure of the edges by simulating random walks on the graph. As a walker proceeds in his random walk from vertex to vertex, he visits some vertices more often than the others. The vertices can thus be ranked according to the probabilities that a walker will arrive at the vertices after such random walk. To simulate such random walk and compute the scores of the vertices, we can represent the graph by its adjacency matrix and compute the fix point of the product of the matrix with itself [15] or approximate the computation of the fix point with PageRank [11] which introduces 'fatigue' to the random walker.

Random walk on graphs has also been used to compute marginalized kernel (or expected similarity) between labeled graphs [12]. The kernel is defined on infinite path count vectors, each path obtained by random walks on the graphs. The marginalized kernel is computed as the inner product of the count vectors averaged over all possible paths. Using random walk algorithm and matrix multiplication till convergence, all possible paths can be accounted for without computing the vector values explicitly [12].

Random walk on graphs has also been used in clustering. Markov clustering (MCL) [4] simulates stochastic flow (or random walks) in the graph. The graph is first represented as stochastic (Markov) matrices where edges between vertices are weighted by the amount of flow between the vertices. MCL algorithm simulates flow using two alternating algebraic operations on the matrices. The flow is eventually separated into different regions, yielding a cluster interpretation of the initial graph.

2.1.1.2 Minimum Spanning Tree Algorithm

Given a connected, weighted, bi-directed graph G = (V, E), a spanning tree of G is a subgraph which is a tree that connects all the vertices in G. The weight of a spanning tree is the sum of the weights of edges in that spanning tree. A minimum spanning tree (MST) of G is a spanning tree whose weight is less than or equal to weight of every other spanning tree of G. More generally, any weighted, bi-directed graph (not necessarily connected) has a minimum spanning forest, which is a union of MSTs of its connected components. Well known algorithms for finding MST are Kruskal's algorithm [16], Borůvka's algorithm [17], and Prim's algorithm [18], which are all greedy algorithms. Faster randomized MST algorithm has been developed in [19].

The randomized MST algorithm uses two important properties of graph, namely the cut and the cycle property [19]. The cut property states that for any proper, non empty subset X of the vertices in the graph, the lightest edge with exactly one end point in X will be in the MST of the graph. The cycle property states that for any cycle C in a graph, the heaviest edge in C will not be in the MST of the graph. In its implementation, the randomized MST algorithm implements the cut property in its Borůvka steps and the cycle property in its random sampling steps.

Like Borůvka's algorithm [17], Borůvka step includes selecting, for each vertex, the minimum weight edge incident to the vertex. Borůvka step then contracts all the selected edges, replacing by a single vertex each connected component defined by the selected edges. All resulting isolated vertices, self-loops, and all except the minimum weight edge between vertices are deleted. All contracted edges are those that satisfy the cut property and are included in the MST of the graph.

The random sampling step is conducted to discard edges that cannot be in the MST of the graph based on the notion of *heavy* and *light* edges. Given a graph G = (V, E) with weighted edges (i.e. c (x, y) is the weight of an edge (x, y) $\in$E), let T be a MST of a random subgraph of G and T (x, y) be the path connecting the vertex x and y in T. Let c_T (x, y) be the maximum weight of edges in T (x, y). An edge (x, y) $\in$E is *T-heavy* if c (x, y) > c_T (x, y) and it is *T-light* otherwise.

According to the cycle property, *T-heavy* edges cannot be in the MST of the graph and are discarded.

The randomized MST algorithm alternates the application of Borůvka step and random sampling step on the graph to reduce the space of the problem and to recursively find the MST of the graph [19]. This randomized MST runs in expected time linear to the number of edges in the graph, i.e. $\Theta(M)$ where $M = |E|$, the number of edges in G.

2.1.2 Clustering Algorithms

In this section we review some state of the art clustering algorithms. Several algorithms have been proposed for clustering. They can be grouped into the following categories: partitioning algorithms, hierarchical algorithms, and graph-based algorithms.

2.1.2.1 Partitioning Clustering Algorithms

K-means clustering algorithm [20] divides the set of vertices of a graph into K clusters by first choosing randomly K seeds or candidate centroids. It then assigns each vertex to the cluster whose centroid is the closest. K-means iteratively re-computes the position of the exact centroid based on the current members of each cluster, and reassigns vertices to the cluster with the closest centroid until a halting criterion is met (e.g. centroids no longer move). The number of clusters K is defined by user a priori and does not change.

K-means clustering attempts to maximize intra-cluster similarity by minimizing the average distance between vertices and their centroids or, equivalently, by maximizing the average similarity between vertices and their centroids. K-means converges because the average distance between vertices and their centroids monotonically decreases at each iteration. First, the average distance between vertices and their centroids decreases in the re-assignment step since each vertex is assigned to the closest centroid. Second, this value decreases in the recomputation step

because the new centroid is the vertex for which this average distance between vertices and the centroid reaches its minimum.

A variant of K-means efficient for document clustering is K-medoids [21]. It uses medoids, i.e. document vectors in the cluster that are closest to the centroid, instead of centroids. Since medoids are sparse document vectors, distance computations are fast. In our experiments, we have used K-medoids to compare with the performance of our proposed algorithms.

2.1.2.2 Hierarchical Clustering Algorithms

Hierarchical algorithms [3] can be categorized into agglomerative and divisive ones. Agglomerative algorithms treat each vertex as a separate cluster, and iteratively merge clusters that have the greatest similarity from each other until all the clusters are grouped into one. The objective function of hierarchical clustering is intra-cluster similarity; i.e. greatest similarity at each merger. Divisive algorithms start with all vertices in one cluster, and subdivide it into smaller clusters.

Hierarchical clustering is categorized into average-, complete-, and single-link. Average-link merges two clusters that have greatest average similarity between any two members. Complete-link merges two clusters that have greatest minimum similarity between any two members. Single-link merges two clusters that have greatest maximum similarity between any two members. Hierarchical clustering can also be considered graph-based clustering algorithms. Single-link clusters are subgraphs of the MST of the data and complete-link clusters are maximal complete subgraphs of the data [1].

Hierarchical clustering results in a hierarchy of clusters. To get disjoint clusters, the hierarchy is cut at some level of similarity x so that the resulting clusters will each have a similarity of at least x. Another way is to cut the hierarchy where the gap between two successive merging of clusters is largest. Such large gaps indicate that adding one more cluster decreases the quality of

clustering significantly; cutting before this steep decrease occurs is viewed as optimal. Another way is to stop building the hierarchy when K clusters are already formed (K being the number of clusters defined by user a priori). This is equivalent to cutting K-1 links between clusters, starting from the top of the hierarchy.

2.1.2.3 Graph-based Clustering Algorithms

Graph-based algorithms for clustering create clusters by cutting or removing edges that are deemed unimportant according to some measurement. We have studied several graph-based clustering algorithms which include Minimum Spanning Tree (MST) clustering, Chameleon, Markov clustering, and Star clustering.

2.1.2.3.1 MST Clustering

Zahn's MST clustering algorithm [2] is a well known graph-based algorithm for clustering [1]. The implementation of Zahn's algorithm starts by finding a minimum spanning tree in the graph and then removes inconsistent edges from the MST to create clusters [22]. Inconsistent edges are defined as edges whose distances are significantly larger (e.g. *c* times larger) than the average distance of the nearby edges in the MST; where *c* is a measure of significance and is a user-defined constant. The objective function of Zahn's algorithm is inter-cluster sparsity; i.e. maximum distances in-between clusters. Zahn's algorithm requires constructing the complete MST to determine inconsistent edges.

More recently, in an application to clustering Chinese news documents, a version of MST clustering that removes exactly K-1 inconsistent edges is used in [23]. The method produces K clusters of high precision and better performance when compared to K-means and single-link. However, the value of K (number of clusters) needs to be determined a priori.

Grygorash et al., in [24], propose two variants of MST clustering: HEMST and MSDR. HEMST assumes that the desired number of clusters is given in advance while MSDR does not. MSDR

starts by constructing MST in the graph. Next, it computes the standard deviation of the edges in the MST, and removes an edge to obtain a set of disjoint sub-trees such that the overall standard deviation reduction is maximized. This edge removing step is done repeatedly to create more disjoint sub-trees until overall standard deviation reduction is within a threshold. Similar to Zahn's, MSDR requires constructing the complete MST in the graph to compute the standard deviation.

He and Chen, in [25], propose an automatic detection of threshold for MST clustering. The method is analogous to finding the cut in hierarchical clustering where the gap between two successive combination similarities is largest. Similar to Zahn's and single-link, this algorithm constructs the complete MST before cutting.

2.1.2.3.2 Chameleon

Chameleon [26] is a graph-based clustering algorithm that stems from the need for dynamic decisions in place of static parameters. Chameleon first constructs a sparse graph of K-nearest neighbour as an initial threshold. It then uses a min-cut based algorithm to partition this graph. Finally, it iteratively and dynamically merges the sub graphs considering their measures of relative inter-similarity and closeness based respectively, on the sum and average weight of edges in the min-cut of and in-between the clusters. The objective function of Chameleon is inter-cluster sparsity (by using min-cut to partition the graph) and intra-cluster similarity (by using relative inter-similarity and closeness to merge subgraphs). Like hierarchical clustering, Chameleon produces a dendrogram of possible clusters at different levels of granularity. In effect, Chameleon uses three parameters: the number of nearest neighbors, the minimum size of the sub graphs, and the relative weightage of inter-similarity and closeness. Although the authors observed that the parameters have a mild effect on 2D data, the users always need to preset the values of parameters and their effects are unknown for other types of data such as documents (high dimensional data).

In a more recent paper [27], the author of Chameleon combines the idea of agglomerative hierarchical clustering and partitional clustering algorithms to propose a new class of agglomerative algorithm that constrains the agglomeration process using clusters obtained by partitional algorithms. Their experimental results suggest that the combined methods improve the clustering solution.

2.1.2.3.3 Markov Clustering

Markov Clustering (MCL) algorithm is a form of graph-based clustering algorithm that is based on simulation of stochastic flow (or random walks) in graphs [4]. The aim of MCL is to separate the graph into regions with many edges inside and with only a few edges between regions. Once inside such a region, the flow (or a random walker) has little chance to flow out [28]. To do this, the graph is first represented as stochastic (Markov) matrices where edges between vertices indicate the amount of flow between the vertices: i.e. similarity measures or the chance of walking from one vertex to another in the graph. MCL algorithm simulates flow using two alternating simple algebraic operations on the matrices: expansion, which coincides with normal matrix multiplication, and inflation, which is a Hadamard power followed by a diagonal scaling. The expansion process causes flow to spread out and the inflation process represents the contraction of flow: it becoming thicker in regions of higher current and thinner in regions of lower current. The flow is eventually separated into different regions, yielding a cluster interpretation of the initial graph. Like Chameleon, the objective function of MCL is intra-cluster similarity and inter-cluster sparsity. However, MCL is computationally expensive as it involves expensive matrix operations.

MCL does not require an a priori number of expected clusters nor a threshold on the similarity values. However, it requires a fine tuning inflation parameter that influences the coarseness and possibly the quality of the resulting clusters.

2.1.2.3.4 Star Clustering

Star clustering is another graph-based algorithm. The algorithm, first proposed by Aslam et al. in 1998 [13], replaces the NP-complete computation of a vertex-cover by cliques by the greedy, simple and inexpensive computation of star shaped dense sub graphs. Star clustering starts by removing edges whose weight is heavier than certain threshold. It then assigns each vertex to its adjacent star center, which is a vertex with equal or higher degree. Each star center and its adjacent vertices is a cluster.

Unlike K-means, Star clustering does not require the indication of an a priori number of clusters. It also allows the clusters produced to overlap. This is a generally desirable feature in information retrieval applications. Star clustering also analytically guarantees a lower bound on the topic similarity between the documents in each cluster and computes more accurate clusters than either the older single-link [29] or average-link [30] hierarchical clustering methods. The objective function of Star is both intra-cluster similarity (by selecting star centers with high degree to potentially maximize intra-cluster similarity) and inter-cluster sparsity (by removing edges whose weights are above certain threshold). However, Star clustering requires the value of threshold to be determined a priori.

Star algorithm has drawbacks that the Extended Star algorithm [31] by Gil et al. proposes to solve. The first drawback is the clusters produced may not be unique. When there are several vertices of the same highest degree, the algorithm arbitrarily chooses one as Star. The second drawback they claim is because no two Star centers can be adjacent to one another; the algorithm can produce illogical clusters [31]. The extended Star algorithm addresses these issues by choosing Star centers independently of document order. It uses complement degree of a vertex, which is the degree of the vertex only taking into account its adjacent vertices not yet included in any cluster. Extended Star algorithm also considers a new notion of Star center where two Star centers are allowed to be adjacent to one another. Extended Star has two versions: unrestricted

and restricted. In the restricted version, only vertices not yet included in any cluster can be Star centers.

On-line Star algorithm [13] supports insertion and deletion of vertices from the graph. When a new vertex is inserted into (or deleted from) the graph, new Stars may need to be created and existing Stars may need to be destroyed. On-line Star algorithm maintains a queue containing all satellite vertices that have the possibility of being 'promoted' into Star centers. Vertices in the queue are processed in order of their degree (from highest to lowest). When an enqueued satellite is promoted to Star center, one or more existing Stars may be destroyed; creating new satellites that have the possibility of being promoted. These satellites are put into the queue and the process repeats.

Section 2.2 Applications

In this research, we explore the application of our proposed methods on the domains of document clustering, k-member clustering for privacy-protection, clustering for part-of-speech tagging, opinion mining and time series correlation.

2.2.1 Document Clustering

Clustering of documents has been investigated for use in various areas of digital libraries, data mining and information retrieval. Initially, clustering of documents was investigated to improve the precision and recall of information retrieval systems [5, 6]. More recently, document clustering has also been used in the field of digital libraries to search a collection of documents [7] or to organize the results returned by search engines in response to user's query [8].

2.2.2 k-member Clustering

One recent application of clustering is data privacy [32]. In a relational database, k-anonymity transforms data to ensure that attributes that could be used to identify real world entities whose privacy is to be protected (quasi-identifiers) are indistinguishable from those of at least (k-1)

other records. The optimal solution to k-anonymity problem has been shown to be NP-hard [33, 34]. The authors of [32] propose to find a set of clusters, each containing at least k records whose quasi-identifier value can be generalized so that records in a cluster are indistinguishable from each other. In order to minimize information loss when records in a cluster are generalized, records in the cluster should be as similar to each other as possible, which is the aim of clustering. Since current clustering algorithms are not directly applicable to solving k-anonymity problem (because they do not consider the requirement that each cluster should contain at least k records), [32] formulates a specific clustering problem called k-member clustering problem and presents a k-member greedy algorithm to solve it.

Given a set of n records, the greedy algorithm first picks a record *r* randomly and makes it a cluster. Then, another record *s* is picked and added to *r*'s cluster. *s* is picked such that the information loss in *r*'s cluster is minimal. This step is repeated until *r*'s cluster has size equals to k. Then, another record *r'* which is furthest from *r* and is not yet assigned to any cluster is picked and the clustering process is repeated until there is less then k unassigned records left. The greedy algorithm then iterates over these leftover records and assigns each record into a cluster with respect to which the increment of the information loss is minimal. k-member greedy algorithm is essentially an adaptation of K-means algorithm; but for which k is the number of records in each cluster instead of the number of clusters K. The metrics to measure distance between records and the metrics to measure the cost of information loss are also presented in [32].

2.2.3 Clustering for Part-of-Speech Tagging

The simplest part-of-speech taggers are based on n-gram models [35], where a word is assigned a tag that has the highest conditional probability of occurring together with the preceding n-1 words and their respective tags. N-gram taggers require relatively large tagged training data. Transformation-based tagging [36] is an example of rule-based machine learning that learns the

rules of tagging from a large set of tagged training data. Unlike n-gram or transformation-based tagging, Hidden Markov Models [37] do not require labeled training data but require a lexicon that specifies possible part-of-speech tags for every word.

In [38], the author proposes the first algorithm for tagging words whose part-of-speech properties are unknown. This is very useful especially for language with no or very limited linguistic source or annotated resource. Similarity between two words is first determined using their left and right neighbors. Each word is represented by a feature vector with one dimension for each neighbor; the cosine between these feature vectors determines the similarity between the corresponding words. Using this similarity measure, words are clustered using Buckshot algorithm [7] that first employs hierarchical clustering algorithm to find centroids and then uses these centroids as initial centroids for K-means clustering.

In [39], the authors extend the approach in [38] by considering a broader context for feature vectors. This approach was shown to be superior over all other existing methods with the Brown corpus. The authors in [39] also observe that in an Indonesian language corpus, words with the same affixes tend to be in the same cluster, thus confirming the potential role of morphology and its interaction with syntax in the definition of part-of-speech classes.

2.2.4 Opinion Mining

Opinion mining and sentiment analysis are the generic natural language processing and text mining tasks involved in the processing of documents that express views and reviews in order to identify attitudes. One specific instance of opinion mining is the rating and ranking of items based on textual reviews. Its subtasks consist of determining and quantifying semantic orientation i.e. whether an object expresses a positive or negative opinion. The semantic orientation of an item, the feature of an item or the review for an item is usually aggregated from the semantic orientation of terms in the reviews: words, word senses or words of certain classes.

The semantic orientation of terms is determined using starting set of terms whose semantic orientation is known and the terms' context in the review (collocation and pivot words such as conjunctions and adverbs, for instance), in some corpus (the Web), or in some known ontological resource like WordNet.

The authors of [40] propose to determine the semantic orientation of adjectives in texts. They use conjunctions (e.g. "and", "but") to derive the semantic orientation of adjectives. For instance "and" connects adjectives of the same orientation and "but" connects adjectives of different orientation. A clustering algorithm partitions adjectives into positive and negative clusters based on the conjunctions that links them.

The authors of [41] propose to determine the semantic orientation of word senses. They construct a lexicon called SentiWordNet where each word sense is associated with three scores, an objective score, a positive score and a negative score, to represent its semantic orientation. They use WordNet synsets and lexical relations together with a machine learning classifier to determine the scores. The same authors in [42] use PageRank on WordNet for the same task.

The author of [43] proposes to determine the semantic orientation of reviews by determining and aggregating the semantic orientation of phrases in the reviews that contain adjectives and adverbs. He quantifies the semantic orientation of a phrase using its collocation with positive and negative adjectives and adverbs in Web documents as retrieved by search engines. The review is then classified as "recommended" if the average semantic orientation of its phrases is positive. It is classified as "not recommended" otherwise.

The authors of [44] propose to determine the semantic orientation of features of items (for instance: the battery life of a cell phone, the user friendliness of its menus, etc.). For this they determine and aggregate the semantic orientation of words in reviews. They leverage the observation that opinions with the same semantic orientation are commonly expressed in

consecutive sentences, unless words such as “but”, “however” articulate the successive sentences. If such words appear, the orientation is changed. The semantic orientation of a starting set of words is used to infer the orientations of other words. If the overall score of words expressed on a feature f in the sentence s is positive (resp. negative), then the semantic orientation of the opinion on f in the sentence s is positive (resp. negative). The authors of [44] argue that semantic orientation should be context-dependent. It must capture usage in context. For instance the adjective “sharp” may be positive or negative depending on the item being reviewed.

2.2.5 Time Series Correlation

A time series is a sequence of real numbers, each representing a value at a time point [45]. Some examples of time series include stock prices, currency exchange rates, biomedical measurements such as ECG signals and weather data.

Time series mining finds its applications in many areas, such as in bioinformatics, chemical engineering, biometric, medical applications, weather forecast or financial market. For example, in weather forecast, one may want to find out whether the weather in one area has an effect in another area. In financial market, the changes in company A’s stock price may have affected another company’s stock price if the two stock prices are correlated. Correlation analysis is important to identify potentially interacting pairs or sets of time series across the database. A strongly correlated set of time series indicates the potential change in one series when the dominating series changes.

We define two time series to be correlated when changes in one time series are related to changes in the other time series. We identify changes in a time series as a sequence of its gradients: the sequence of differences between each pair of successive values in the time series. The correlation measure is therefore reduced to that of measuring similarity between the gradient sequences. We define two time series to be positively correlated when they have high similarity in their

corresponding gradient sequences: i.e. when one time series increases (resp. decreases) as the other time series increases (resp. decreases).

Traditionally, the Euclidean distance [46, 47] is used as a measure for computing the distance (or similarity) between two sequences that are properly aligned in time. The total distance is aggregated together from the corresponding pairwise distance. However, not all sequences can be properly aligned in time. One sequence may have a delayed response to the changes in the other sequence, or perhaps a delayed response to a common stimulus that affects both sequences. The response of one sequence to the changes in another sequence or a common stimulus may also be "smeared" in time. Lagged correlation is the term we use to refer to the correlation between two time series that are shifted in time relative to one another. Computing lagged correlation between two time series by properly aligning their gradient sequences in time may be inadequate because it does not take into account the shift (or misalignment) in time.

To measure similarity between two sequences that allows for misaligned mapping, Longest Common Subsequence (LCS) can be used. LCS is based on edit distance to measure the similarity of two sequences. However, LCS has its own drawback. It can only measure global similarity but fail to find partial similarity (cf. figure 1). The result found is always the longest common subsequence between two complete sequences [48].

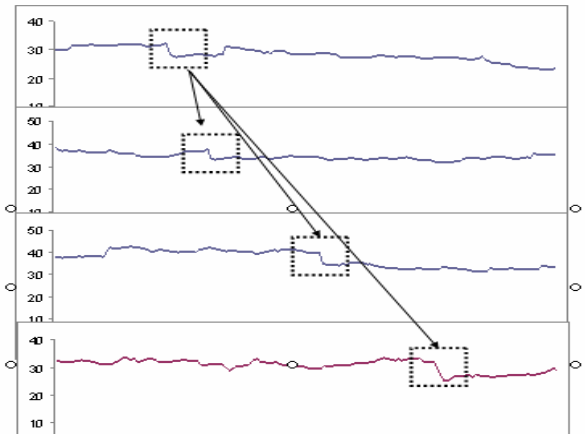

Fig. 1. Partial Similarity between Sequences

CHAPTER 3. PROPOSED METHODS

In this research, we study graphs, clustering and their applications. In this chapter we present our proposed methods.

Section 3.1 Graphs

Based on our study of graph algorithms – random walk and minimum spanning tree algorithm – we propose two novel methods that apply random walk algorithm in opinion mining and in computing time series correlation. We also propose a novel variant of randomized minimum spanning tree algorithm that is based on random sampling alone.

From our study of graph-based clustering algorithms, we propose novel variants of Star clustering algorithm and two novel graph-based clustering algorithms: MST-Sim and Ricochet.

We explore various techniques, including random walk algorithm, for identifying centroids (Star centers) in our variants of Star clustering algorithm. We explore minimum spanning tree algorithm in our MST-Sim clustering, adding to it intra- and inter-cluster similarity metrics that have basis in graph theory. Ricochet uses the results of our study on Star clustering to identify centroids and the results of our study on minimum spanning tree algorithm to identify edges to merge clusters. Ricochet also uses the idea of vertices reassignment to clusters in K-means to reach its termination condition. Ricochet is unconstrained; it does not require any parameters to be defined a priori. This is unlike Star clustering (that needs threshold to be defined a priori) or K-means (that needs the value of K to be defined a priori).

3.1.1 Graph Algorithms

3.1.1.1 Random Walk

We propose two novel methods based on random walk on graphs and apply them in the domain of opinion mining – to rank objects based on opinions in their reviews, and in the domain of time

series – to compute correlation between two time series. These proposed methods are detailed in the applications section 3.2.4 and 3.2.5 respectively.

3.1.1.2 A Variant of Randomized MST Algorithm

Here we present our proposed variant of randomized MST algorithm. From our study of the randomized MST algorithm [19], we observe several drawbacks of the algorithm. Firstly, the decision on how many Borůvka's steps applied at each recursive call of the algorithm is not clear. [19] Applies 2 Borůvka's steps at each recursive call of the algorithm while [49] applies 4 Borůvka's steps at each recursive call of the algorithm. If there are too many Borůvka's steps, the randomized MST algorithm is no different than Borůvka's algorithm; too few and the randomized algorithm will not be able to run in linear time. It is not clear what the best number is for the application of Borůvka's steps in the algorithm.

Secondly, given a graph G and a random subgraph G1 obtained from G, the algorithm recursively finds the MST T of G1 and use T to identify *T-heavy* and *T-light* edges in G. Based on the cycle property however; any random tree T in the graph (not necessarily MST) can be used to identify *T-heavy* edges. The choice of the tree is irrelevant to the result, i.e. the MST of G. By recursively finding the MST of G1, the algorithm may incur additional cost when they could have selected and used any tree in G. Given any tree T in the graph, the linear time MST verification algorithm [50] can be used to identify *T-heavy* edges in the graph.

Given a tree T from G, the verification algorithm [50] first builds a Borůvka tree B by applying Borůvka algorithm on T. Since T is a tree with N-1 edges, B can be constructed in O (N) time where N is the number of vertices in G. The property of Borůvka tree B necessitates that the weight of the heaviest edge in the path between two vertices x and y in T equals the weight of the heaviest edge in the path between x and y in B, which can be found in unit time once B is

constructed [50]. An edge (x, y) with weight c (x, y) in G is *T-heavy* (and therefore is not in the MST of G) if the heaviest edge in the path between x and y in B is lighter than c (x, y).

We abandon the idea of Borůvka steps application and the recursive discovery of MST in random subgraphs to propose a variant of randomized MST algorithm.

Given a graph G = (V, E) where N = |V| is the number of vertices in G and M = |E| is the number of edges in G, our proposed algorithm first selects a random subgraph T with N-1 edges from G and incrementally builds MST of G from T. To do this, the algorithm iterates through all the edges in G and discards or keeps edges accordingly. The algorithm uses T with the linear time MST verification algorithm [50] to identify and discard *T-heavy* edges. Based on the cycle property, these *T-heavy* edges are not in the MST of G and can be discarded. The algorithm also discards edges that are not selected during the application of Borůvka algorithm on T [50]. Based on the cut property, these edges are not in the MST of G and can be discarded. If an edge *e* = (x, y) that is not in the Borůvka tree is *T-light*, *e* is added to the Borůvka tree. Correspondingly, edge *e*', the heaviest edge in the path between x and y in the Borůvka tree is discarded because *e*' is now *T-heavy*. Once all the edges are examined, the remaining edges in G that are not discarded are the MST of G. In total, our algorithm only does one scan through the edges in G to find MST.

Our proposed algorithm does not need to decide how many Borůvka steps to apply and it fully utilizes the cycle and cut property of graphs. The algorithm is detailed in figure 2. The complexity of the algorithm is at most O (MN) because it needs to iterate through at most M edges in G. Each iteration may cost a re-construction of the Borůvka tree when an edge is added or removed from the tree. A construction of the Borůvka tree costs at most O (N) since there are only N-1 edges in the tree [50]. Hence the complexity of the algorithm is at most O (MN).

The complexity of our algorithm depends on its selection of the initial random subgraph T. The algorithm can run faster if it selects a good T from G, i.e. T which is close to the MST of G.

Algorithm: A variant of randomized MST algorithm
Input: Graph G = (V, E) where N = |V| (the number of vertices) and M = |E| (the number of edges)
Output: MST of G

1. Obtain a subgraph T of G by randomly including N-1 edges from G
2. Construct a Borůvka tree B from T
3. Identify edges in T that are not selected to construct B, delete these edges from G and T
4. Using a linear-time verification algorithm, identify T-heavy edges in G and delete them from G and T, i.e. for each edge *e* = (x, y) in G:
    a. If *e* is T-heavy, delete it from G and T
    b. If *e* is T-light:
        i. identify the heaviest edge *e*' in the path between x and y in B that makes *e* T-light and remove *e*' from G and T
        ii. add *e* to T
        iii. reconstruct the Borůvka tree B from T
        iv. delete edges that are not selected to construct B from G and T
    c. If *e* connects disconnected component in T:
        i. add *e* to T
        ii. reconstruct the Borůvka tree B from T
        iii. delete edges that are not selected to construct B from G and T
5. Return remaining edges in G

Fig. 2. A Variant of Randomized MST Algorithm

3.1.2 Graph-based Clustering Algorithms

In this research, we study existing graph-based clustering algorithms and evaluate their choice of similarity metrics, empirically and theoretically using the basis of graph theory; and using different measure of threshold when applicable. We have performed a study on Star clustering and MST clustering. Combining the ideas from Star clustering, MST, and K-means, we propose a novel family of graph-based clustering algorithms called Ricochet that does not require any parameters to be defined a priori.

3.1.2.1 Star Clustering [51]

To produce reliable document clusters of similarity σ (i.e. clusters where documents have pair-wise similarities of at least σ, where σ is a user-defined threshold), the Star algorithm starts by representing the document collection by its σ-similarity graph. A σ-similarity graph is an undirected, weighted graph where vertices correspond to documents and there is an edge from vertex v_i to vertex v_j if their cosine similarity in a vector space is greater than or equal to σ. Star clustering formalizes clustering by performing a minimum clique cover with maximal cliques on

this σ-similarity graph (where the cover is a vertex cover). Since covering by cliques is an NP-complete problem, Star clustering approximates a clique cover greedily by dense sub-graphs that are star shaped.

A star shaped sub-graph of m + 1 vertices consisting of a single Star center and m satellite vertices, where there exist edges between the Star center and each satellite vertex. Star clustering guarantees pair-wise similarity of at least σ between the Star and each of the satellite vertices. However, it does not guarantee such similarity between satellite vertices. By investigating the geometry of the vector space model, Aslam et al. derive a lower bound on the similarity between satellite vertices and predict that the pair-wise similarity between satellite vertices in a Star-shaped sub-graph is high. In their derivation of expected similarity between satellite vertices in a Star cluster [13], Aslam et al. show that the lower bound of similarity cos ($\gamma_{i,j}$) between two satellite vertices v_i and v_j in a Star cluster is such that:

$$\cos(\gamma_{i,j}) \geq \cos(\alpha_i)\cos(\alpha_j) + (\sigma / \sigma + 1)\sin(\alpha_i)\sin(\alpha_j) \qquad (2)$$

where cos (αi) is the similarity between the Star center v and satellite v_i and cos (αj) is the similarity between the Star center v and satellite v_j. They also show empirically that the right hand side of inequality (2) above is a good estimate of its left hand side.

The off-line Star clustering algorithm (for static data) starts by sorting the vertices by degree. Then, it scans the sorted vertices from highest to lowest degree as a greedy search for Star centers. Only vertices that do not yet belong to a Star can become Star centers. Once a new Star center v is selected, its marked bits are set, and for all vertices w adjacent to v, w.marked bit is set. Only one scan of the sorted vertices is needed to determine all Star centers. Upon termination (i.e. when all vertices have their marked bits set) these conditions must be met: (1) the set of Star centers are the Star cover of the graph, (2) a Star center is not adjacent to any other Star center, and (3) every satellite vertex is adjacent to at least one center vertex of equal or higher degree.

The algorithm has a running time of $\Theta (V + E\sigma)$ where V is the set of vertices and $E\sigma$ edges in the σ-similarity graph $G\sigma$ [13].

Our work on Star clustering [51] is motivated by the suspicion that degree is not the best similarity metrics for the selection of Star centers because it does not make use of the very derivation that produced the inequality (2). We believe Star centers should be selected with a metric that tries to maximize the average intra-cluster similarity, i.e. metric that takes into consideration the weight of the edges at any given vertex rather than just its degree. The derivation (2) can therefore be used to estimate the average intra-cluster similarity. For a cluster of n vertices and center v, the average intra-cluster similarity is:

$$(\Sigma_{(vi\,,\,vj)\in\, v.adj \times v.adj}\ (\cos(\gamma_{i,j}))\ ) / n^2 \geq ((\Sigma_{vi\,\in\, v.adj} \cos(\alpha_i))^2 + (\sigma / \sigma + 1)\ (\Sigma_{vi\,\in\, v.adj} \sin(\alpha_i))^2)/n^2 \qquad (3)$$

where $\gamma_{i,\,j}$ is the angle between vertices v_i and v_j, α_i is the angle between v and vertex v_i, and where v.adj is the set of vertices adjacent to v in $G\sigma$ (i.e. the vertices in the cluster). This is computed on the pruned graph, i.e. after edges with weights lower that σ has been removed from the graph. For each vertex v in $G\sigma$, we let:

$$lb\ (v) = ((\Sigma_{vi\,\in\, v.adj} \cos(\alpha_i))^2 + (\sigma / \sigma + 1)\ (\Sigma_{vi\,\in\, v.adj} \sin(\alpha_i))^2)/n^2 \qquad (4)$$

We call the metrics lb(v), the lower bound. This is the theoretical lower bound on the actual average intra-cluster similarity when v is Star center and v.adj are its satellites. We introduce several similarity metrics that tries to maximize this lower bound of intra-cluster similarity with the hope of improving the actual intra-cluster similarity.

Among the metrics that we introduce are (a) Markov Stationary Distributions that uses the idea of random walk to find important vertices in the graph that are potentially good candidates for star centers, (b) Lower Bound that chooses vertices which maximize the value of lower bound derived in (4) as star centers, (c) Average that chooses vertices which maximize the function *ave(v)* as star centers (where $ave(v) = \Sigma_{\forall vi\,\in\, v.adj} \cos(\alpha_i)/$ degree(v); and α_i is the angle between v

and vertex vi), and (d) Sum that chooses vertices which maximize the function *sum(v)* as star centers (where sum(v) = $\Sigma_{\forall vi \in v.adj} \cos(\alpha_i)$; where α_i is the angle between v and vertex v_i).

Note that our function *ave(v)* is the square-root of the first term in the lower bound *lb(v)* metric. Thus we expect *ave(v)* will be a good approximation for *lb(v)*. Coincidentally, *ave(v)* which aims to maximize the average similarity between a Star center and its satellites is related to the idea used by K-means algorithm whereby, at each iteration, the algorithm tries to maximize the average similarity between centroids and their vertices.

Incorporating these metrics in off-line and on-line Star algorithms, we sort vertices by sum and average and pick unmarked vertex with highest sum and average to be the new Star center, respectively. In the remainder we refer to Star algorithm in which Star centers are determined using the sum and average as the Star-sum and Star-ave, respectively.

We incorporate the lower bound, average and sum metrics in the original and extended Star algorithms by using *lb(v)*, *sum(v)* and *ave(v)* in the place of *degree(v)* and by defining and using complement lower bound *CL(v)*, complement sum *CS(v)*, and complement average *CA(v)* in the place of complement degree for the extended Star algorithm *CD(v)*, respectively. We define (where Clu is the set of vertices already clustered):

$$CL(v) = ((\Sigma_{vi \in v.adj \setminus Clu} \cos(\alpha_i))^2 + (\sigma / \sigma + 1)(\Sigma_{vi \in v.adj \setminus Clu} \sin(\alpha_i))^2)/n^2 \qquad (5)$$

$$CS(v) = \Sigma_{\forall vi \in v.adj \setminus Clu} \cos(\alpha_i) \qquad (6)$$

$$CA(v) = \Sigma_{\forall vi \in v.adj \setminus Clu} \cos(\alpha_i)/ CD(v) \qquad (7)$$

We integrate the above metrics in the Star algorithm and its variants to produce the following extensions: (1) Star-lb: the off-line star algorithm with *lb(v)* metrics; (2) Star-sum: the off-line star algorithm with *sum(v)* metrics; (3) Star-ave: the off-line star algorithm with *ave(v)* metrics; (4) Star-markov: the off-line star algorithm with Markov stationary distribution metrics; (5) Star-extended-sum-(r): the off-line restricted extended star algorithm with *CS(v)* metrics; (6) Star-

extended-ave-(r): the off-line restricted extended star algorithm with *CA(v)* metrics; (7) Star-extended-sum-(u): the off-line unrestricted extended star algorithm with *CS(v)* metrics; (8) Star-extended-ave-(u): the off-line unrestricted extended star algorithm with *CA(v)* metrics; (9) Star-online-sum: the on-line star algorithm with *sum(v)* metrics; (10) Star-online-ave: the on-line star algorithm with *ave(v)* metrics.

For the sake of simplicity and concision, in the experimental section we only show results for lower bound in the original Star algorithm (we do not evaluate Star-extended-lb-(u) and Star-extended-lb-(r), the off-line unrestricted and restricted extended star with lower bound metrics, nor Star-online-lb: the on-line star algorithm with lower bound metrics).

3.1.2.2 MST-Sim Clustering

In this research, we propose a novel MST-based graph-based clustering algorithms, called MST-Sim, that draws inspiration from Kruskal's algorithm and Borůvka's algorithm for finding minimum spanning tree (MST) in the graph [16, 17]. A spanning tree, by definition, is an acyclic sub graph of graph G that contains all the vertices in G. In a weighted graph (where each edge is weighted by the distance between its end vertices: the smaller the distance, the more similar the vertices), a minimum spanning tree is the spanning tree with minimum weight in that graph.

There are several well known algorithms for finding MST in a weighted graph. Kruskal's algorithm [16] sorts edges in ascending order of their weight. Edges and the vertices they connect are added to the minimum spanning tree in this order. The algorithm terminates when all vertices have been connected into a single component. Borůvka's algorithm [17] scans the set of vertices and connects each vertex to the vertex or sub-tree with the lightest edge. This step is iterated for sub-trees until all vertices are connected into a single component. Prim's algorithm [18] starts with an arbitrary vertex and repeatedly connects a new adjacent vertex with the lightest edge.

Reverse-delete algorithm [52] deletes edges from the original graph in a descending order of their weight and that do not disconnect the graph until the result is a tree.

Our proposed family of MST-Sim clustering algorithms is different in several aspects from the existing MST clustering algorithms. Firstly, our algorithms do not require any input information such as desired number of clusters or threshold to be provided before clustering. At most they use an inflation parameter that gives a fine tuning capability for effectiveness. Secondly, our algorithms find clusters while they are constructing the MST rather than a posteriori. Thirdly, our simple algorithms consider both inter-cluster sparsity – edges in-between clusters and intra-cluster density – edges within clusters when constructing clusters. They leverage the following properties of Kruskal's and Borůvka's algorithms

Theorem 1: At any point in the execution of Kruskal's algorithm, the intermediary result is a spanning forest of k sub-trees with maximum spacing. Spacing is defined as the sum of the $k \times (k-1)/2$ minimum weights edges between sub-trees.

Proof: *Let T= {T1, T2 …, Tk} be an intermediary set of sub-trees in an edge-weighted graph computed by Kruskal' algorithm. T is a spanning forest of the initial graph because it contains every vertex. Let w be the weight of the next edge to be considered. w is the lightest among all unprocessed edges. Hence the spacing between the sub-trees in T is heavier or equal to w. For any different spanning forest of k sub-trees T' = {T1', T2'…, Tk'}, T' ≠ T, there exists at least two vertices vi and vj that were adjacent in the same sub-tree Tl ($1 \leq l \leq k$) in T but are now in different sub-trees, Ti' and Tj' ($1 \leq i, j \leq k, i \neq j$) in T'. Since vi and vj are in Tl in T, there exists an edge connecting them with weight lighter or equal to w. The spacing of T' is hence no heavier than the spacing in T. Therefore the clustering T has maximum spacing between clusters.* □

A similar result can be established for Borůvka's algorithm.

Theorem 2: (from [53]) if T is a MST for graph G and X, Y are two vertices of G, then the unique path in T from X to Y is a minimax path from X to Y. A minimax path minimizes over all paths the sum of the maximum weight of edges in the path. Minimax paths in the MST connect two vertices X and Y belonging to a tight cluster without straying outside the cluster [2].

Theorem 1 and theorem 2 suggest that the minimum spanning sub-trees created during the execution of Kruskal's and Borůvka's algorithms may exhibit good inter- and intra-cluster similarities; as the way these sub-trees are created aims to maximize the minimum weights of edges between sub-trees and minimize the maximum weights of edges within sub-tree.

3.1.2.2.1 General MST-Sim Algorithm

We can now outline the principle of the MST-Sim algorithms that we propose. During the execution of either Kruskal's or Borůvka's algorithms, the merging of two, non singleton sub-trees is decided based on metrics measuring inter- and intra-cluster similarity. We study several metrics proposed in [54]. We call these algorithms *MST-Sim name-of-metrics (value-of-inflation-parameter)* and indicate the algorithm used, *Kruskal* or *Borůvka*, when ambiguous.

By performing a more informed decision on merging, *MST-Sim* proposes to achieve a balance between recall and precision of clusters. To do so, when *MST-Sim* encounters an edge connecting two vertices from different sub-trees, using a similarity metric *Z*, it first calculates the inter-cluster similarity (based on the weights of edges between the clusters) and intra-cluster similarity (based on the weights of edges within the clusters). *MST-Sim* then merges the two clusters only if their inter-cluster similarity is bigger than either one of their intra-cluster similarities times an inflation parameter *c*. In our experiments we vary this inflation parameter *c* to tune and evaluate its effect on the clustering results. Figure 3 is the pseudo-code for *MST-Sim Z(c) Kruskal,* and figure 4 the pseudo-code for *MST-Sim Z(c) Borůvka*.

Algorithm: MST-Sim Z(c) Kruskal ( )

Let *E* be the set of edges in the graph

- Sort *E* from lightest to heaviest
- While there are still vertices not assigned to any sub-tree and |*E*| is not empty, take the next lightest edge *e* from *E* (say *e* connects vertices *A* and *B*)
  - If *A* and *B* belong to different sub-trees, merge the two sub-trees if their inter-cluster similarity measured by *Z* is bigger than *c* times either one of their intra-cluster similarities measured by *Z*
  - If *A* belongs to a sub-tree and *B* is not, include *B* in *A*'s sub-tree. Similarly, If *B* belongs to a sub-tree and *A* is not, include *A* in *B*'s sub-tree
  - If both *A* and *B* not yet assigned to any sub-tree, create a new sub-tree that includes both *A* and *B*
- Output the sub-trees as clusters

Fig. 3. Pseudocode of MST-Sim Kruskal

Algorithm: MST-Sim Z(c) Borůvka ( )

Let *V* be the set of vertices in the graph
countOfMerging = 1
mergingAllowed = false
While (countOfMerging>0)

- countOfMerging = 0
- For each vertex *v* in *V* consider its lightest weight, not yet processed edge *e* (say *e* connects *v* to *u*)
  - If *v* and *u* belong to different sub-trees, merge the two sub-trees if their inter-cluster similarity measured by *Z* is bigger than *c* times either one of their intra-cluster similarities measured by *Z* and if mergingAllowed == true. If there is merging of clusters, countOfMerging++
  - If *v* belongs to a sub-tree and *u* is not, include *u* in *v*'s sub-tree. Similarly, If *u* belongs to a sub-tree and *v* is not, include *v* in *u*'s sub-tree
  - If both *v* and *u* not yet assigned to any sub-tree, create a new sub-tree that includes both *v* and *u*
- If mergingAllowed == false, countOfMerging = 1
- mergingAllowed = true

Output the sub-trees as clusters

Fig. 4. Pseudocode of MST-Sim Borůvka

MST-Sim Kruskal needs to investigate all edges within and between clusters before deciding on a merge. Letting |*E*| to be the number of edges in the graph, *MST-Sim Kruskal* has the complexity of O (|*E*| log |*E*|) for sorting the edges in the graph and O ($|E|^2$) for clustering in the worst-case scenario (i.e. in the worst case it needs to iterate through all |*E*| edges in the graph, each time examining at most |*E*| edges). Therefore the algorithm's complexity is O ($|E|^2$).

The complexity of *MST-Sim Borůvka* is O ($log(V)|V|^2$), where $|V|$ is the number of vertices in the graph (i.e. the algorithm does at most $log(V)$ iterations, each time scanning over $|V|$ vertices; at each vertex checking at most $|V|$ edges). The number of iterations is at most $log(V)$ because each iteration reduces number of sub-trees by a factor of 2. Although potentially faster for dense graphs such as similarity clustering, *MST-Sim Borůvka's* may be sensitive to the order of considering vertices.

3.1.2.2.2 Similarity Metrics

We study three similarity metrics to measure intra and inter-cluster similarities. The metrics are *Eta* index, *Beta* index, and *Diameter* index [54].

In line with the incremental nature of our algorithms and to ensure that our algorithms require only local information; we only use edges that are already processed by the algorithm, (i.e. the potential MST edges) instead of using all the edges in the graph to compute intra- and inter-cluster similarities.

For *Eta* index, intra-cluster similarity of a sub-tree is measured as the average of the inverse weights of edges that lie within the sub-tree. Inter-cluster similarity between two sub-trees is measured as the average of the inverse weights of edges that lie in-between the two sub-trees.

$$Eta\text{-}intra\ (T_i) = \Sigma\ (1\ /\ \text{weight}\ (e_{ix}))\ /\ |E_i| \qquad (8)$$

For all e_{ix} ε E_i, where E_i is the list of edges that are already processed by the algorithm and that lie within sub-tree T_i.

$$Eta\text{-}inter\ (Ti,\ T_j) = \Sigma\ (1\ /\ \text{weight}\ (e_{bx}))\ /\ |E_b| \qquad (9)$$

For all e_{bx} ε E_b, where E_b is the list of edges that are already processed by the algorithm and that lie in-between sub-tree T_i and T_j.

For *Beta* index, intra-cluster similarity of a sub-tree is measured as the sum of the inverse weights of edges that lie within the sub-tree divided by the number of vertices in that sub-tree.

Inter-cluster similarity between two sub-trees is measured as half the sum of the inverse weights of edges that lie in-between the sub-trees (here we treat the two sub-trees as two vertices connected by edges in-between).

$$\textit{Beta-intra } (T_i) = \Sigma \ (1 \ / \ \text{weight} \ (e_{ix})) \ / \ |V_i| \qquad (10)$$

For all $e_{ix} \ \varepsilon \ E_i$, where E_i is the list of edges that are already processed by the algorithm and that lie within sub-tree T_i, and V_i is the vertices in T_i.

$$\textit{Beta-inter } (Ti, \ T_j) = \Sigma \ (1 \ / \ \text{weight} \ (e_{bx})) \ / \ 2 \qquad (11)$$

For all $e_{bx} \ \varepsilon \ E_b$, where E_b is the list of edges that are already processed by the algorithm and that lie in-between sub-tree T_i and T_j.

Diameter index measures the length of the shortest path between the most distant vertices of a graph [54]. In other words, it computes the maximum over all the minimum weight paths between any two vertices in the graph. To compute the diameter of a sub-tree while building the MST in the graph (and without incurring further cost), we approximate the diameter of a sub-tree as the sum of weights of MST edges in the sub-tree. Since every path connecting two vertices in MST contains minimum weight edges, the sum of weights of MST edges in the sub-tree is an upper bound to the diameter of the sub-tree.

In an edge-weighted graph where similarity between vertices is inversely proportional to the weight of edges between the vertices, we measure intra-cluster similarity of a sub-tree (using concept from Diameter index) as the sum of the inverse weights of edges that lie within the sub-tree.

$$\textit{Diameter-intra } (T_i) = \Sigma \ (1 \ / \ \text{weight} \ (e_{ix})) \qquad (12)$$

For all $e_{ix} \ \varepsilon \ E_i$, where E_i is the list of edges that are already processed by the algorithm and that lie within sub-tree T_i.

$$\textit{Diameter-inter } (Ti, \ T_j) = \Sigma \ (1 \ / \ \text{weight} \ (e_{bx})) \qquad (13)$$

For all e_{bx} ε E_b, where E_b is the list of edges that are already processed by the algorithm and that lie in-between sub-tree T_i and T_j.

These similarity metrics are used to measure intra-cluster vs. inter-cluster similarities between two sub-trees that are to be merged. If the inter-cluster similarity is bigger than either one of their intra-cluster similarities times an inflation parameter, the two sub-trees are merged. We vary this parameter in our experiment to see how it influences effectiveness.

3.1.2.2.3 Online Clustering

Our family of MST clustering is easily adaptable to an on-line variant. When a new vertex *v* is added to the graph, the weights of edges connecting *v* to vertices already in the graph are computed. Inline with our algorithm, *v* is included in the cluster to which it is connected by an edge with the lightest weight. The edge is added to the MST edges in the graph. O(V) complexity is required whenever a vertex is added to the graph; V is number of vertices already in the graph. When a vertex *v* is deleted from the graph, MST edge(s) that has vertex *v* at any of its end needs to be deleted. A deletion of MST edges may cause clusters to be broken. Hence, whenever a vertex is deleted from the graph, all existing clusters need to be reviewed. One way to do this is to first delete all the clusters, treat each vertex as a cluster, scan through the remaining MST edges (from the first MST edge to the last MST edge used to build the MST in the graph). If the edge connects two vertices from different clusters, merge the clusters. We do not need to check for intra- or inter-cluster similarities when reconstructing clusters since it was done when we first build the MST. O (E') complexity is required whenever a vertex *v* is deleted from the graph; E' is the number of MST edges.

3.1.2.3 Ricochet: a Family of Unconstrained Graph-based clustering [55]

Unlike Star clustering algorithm that is parameterized by the edge threshold and MST clustering that is parameterized by the inflation parameter, Ricochet is unconstrained. Ricochet algorithms

alternate two phases: the choosing of vertices to be the centroids of clusters and the assignment of vertices to existing clusters. The motivation underlying our work is that: (1) Star clustering algorithm provides a metric of selecting Star centers that are potentially good cluster centroids for maximizing intra-cluster similarity, (2) K-means provides an excellent vertices assignment and reassignment, and a convergence criterion that increases intra-cluster similarity at each iteration, (3) minimum spanning tree algorithm provides a means to select edges to merge clusters that potentially maximizes intra-cluster similarity. By using Star clustering for selecting Star centers, we find potential cluster centroids without having to supply the number of clusters. By using K-means re-assignment of vertices, we update and improve the quality of these clusters and reach a termination condition without having to determine any threshold. By using minimum spanning tree algorithm, we select edges to merge clusters to maximize the clusters' intra-cluster similarities. Hence, similar to Star and Star-ave algorithms [13], Ricochet chooses centroids in descending order of the value of a metric combining degree with the weight of adjacent edges. Similar to K-means, Ricochet assigns and reassigns vertices; and the iterative assignment of vertices is stopped once these conditions are met: (1) no vertex is left unassigned and (2) no vertex is candidate for re-assignment. Similar to minimum spanning tree algorithm, for each centroid, Ricochet selects the unprocessed edge with minimum distance adjacent on the centroid.

The Ricochet family is twofold. In the first Ricochet sub-family, centroids are chosen one after the other ('stones are thrown one by one'). In the second Ricochet sub-family, centroids are chosen at the same time ('stones are thrown together'). We call the former algorithms Sequential Rippling, and the latter Concurrent Rippling. The algorithms in the Sequential Rippling, because of the way they select centroids and assign or re-assign vertices, are intrinsically hard clustering algorithms, i.e. they produce disjoint clusters. The algorithms in the Concurrent Rippling are soft clustering algorithms, i.e. they produce possibly overlapping clusters.

3.1.2.3.1 Sequential Rippling

The algorithms in the Sequential Rippling can be perceived as somewhat straightforward extensions to the K-means clustering. We nevertheless present them here for the purpose of completeness and comparison with the more interesting algorithms we propose in the Concurrent Rippling.

3.1.2.3.1.1 Sequential Rippling (SR)

The first algorithm of the subfamily is call Sequential Rippling (or SR). In this algorithm, *vertices* are ordered in descending order of the average weight of their adjacent edges (later referred to as the weight of a vertex). The vertex with the highest weight is chosen to be the first centroid and a cluster is formed by assigning all other vertices to the cluster of this first centroid. Subsequently, new centroids are chosen one by one from the ordered list of vertices. When a new centroid is added, vertices are re-assigned to a new cluster if they are closer to the new centroid than they were to the centroid of their current cluster (if no vertex is closer to the new centroid, no new cluster is created). If clusters are reduced to singletons during re-assignment, they are assigned to the nearest non-singleton cluster. The algorithm stops when all vertices have been considered.

The pseudocode of the Sequential Rippling algorithm is given in figure 5. The worst case complexity of Sequential Rippling algorithm is O (N^3) because in the worst case the algorithm has to iterate through at most N vertices, each time comparing the distance of N vertices to at most N centroids.

3.1.2.3.1.2 Balanced Sequential Rippling (BSR)

The second algorithm of the subfamily is called Balanced Sequential Rippling (BSR). The difference between BSR and SR is in its choice of subsequent centroid. In order to balance the distribution of centroids in the graph, BSR chooses a next centroid that is both a reasonable

centroid for a new cluster (i.e. has large value of weight) as well as sufficiently far from the previous centroids. Subsequent centroid is chosen to maximize the ratio of its weight to the sum of its similarity to the centroids of already existing clusters. This is a compromise between weight and similarity. We use here the simplest possible formula to achieve such compromise. It could clearly be refined and fine-tuned. As in SR, when a new centroid is added, vertices are re-assigned to a new cluster if they are closer to the new centroid than they were to the centroid of their current cluster. The algorithm terminates when there is no re-assignment of vertices.

The pseudocode of Balanced Sequential Rippling algorithm is given in figure 6. The worst case complexity of Balanced Sequential Rippling algorithm is O (N^3) because in the worst case the algorithm has to iterate through at most N vertices, each time comparing the distance of N vertices to at most N centroids.

Given a Graph G = (V, E). V contains vertices, N = |V|. Each vertex has a weight which is the average similarity between the vertex and its adjacent vertices. E contains edges in G (self-loops removed) with similarity as weights.

Algorithm: SR ( )
Sort V in order of *vertices'* weights
Take the heaviest vertex *v* from V
listCentroid.add (*v*)
Reassign all other vertices to *v*'s cluster
While (V is not empty)
    Take the next heaviest vertex *v* from V
    Reassign vertices which are more similar to *v* than to other centroid
    If there are re-assignments
        listCentroid.add (*v*)
        Reassign singleton clusters to its nearest centroid
For all *i* ε listCentroid return *i* and its associated cluster

Fig. 5. Sequential Rippling Algorithm

```
Algorithm: BSR ( )
Sort V in order of vertices' weights
Take the heaviest vertex v from V
listCentroid.add (v)
Reassign all other vertices to v's cluster
Reassignment = true
While (Reassignment and V is not empty)
    Reassignment = false
    Take a vertex v ∉ listCentroid from V whose ratio of its weight to
     the sum of its similarity  to existing centroids is the maximum
    Reassign vertices which are more similar to v than to other centroid
    If there are re-assignments
        Reassignment = true
        listCentroid.add (v)
        Reassign singleton clusters to its nearest centroid
For all i ε listCentroid return i and its associated cluster
```

Fig. 6. Balanced Sequential Rippling Algorithm

3.1.2.3.2 Concurrent Rippling

Unlike sequential rippling which chooses centroids one after another, concurrent rippling treats all vertices as centroids of their own singleton clusters initially. Then, each centroid concurrently 'ripples its influence' using its adjacent edges to other vertices. As the rippling progresses, some centroids can lose their centroid status (i.e. become non-centroid) as the cluster of smaller weight centroid is 'engulfed' by the cluster of bigger weight centroid

3.1.2.3.2.1 Concurrent Rippling (CR)

The first algorithm of the sub-family is called Concurrent Rippling (CR). In this algorithm, for each vertex, the adjacent *edges* are ordered in descending order of weights. Iteratively, the next heaviest edge is considered. Two cases are possible: (1) if the edge connects a centroid to a non-centroid, the non-centroid is added to the cluster of the centroid (notice that at this point the non-centroid belongs to at least two clusters), (2) if the edge connects two centroids, the cluster of one centroid is assigned to the cluster of the other centroid (i.e. it is 'engulfed' by the other centroid), if and only if its weight is smaller than that of the other centroid. The two clusters are merged and the smaller weight centroid becomes a non-centroid. The algorithm terminates when

the centroids no longer change. The pseudocode of Concurrent Rippling algorithm is given in figure 7.

Making sure that all centroids propagate their ripples at equal speed (lines 8 - 10 of Algorithm: CR () in figure 7), the algorithm requires the sorting of a list whose total size is the square of the number of vertices.

Concurrent Rippling algorithm requires O (N^2logN) complexity to sort the N-1 neighbors of the N vertices. It requires another O (N^2logN) to sort the N^2 number of edges. In the worst case, the algorithm has to iterate through all the N^2 edges. Hence, in the worst case the complexity of the algorithm is O (N^2logN).

3.1.2.3.2.2 Ordered Concurrent Rippling (OCR)

The second algorithm of the sub-family is called Ordered Concurrent Rippling (OCR). In this algorithm, the constant speed of rippling is abandoned to be approximated by a simple ordering of adjacent edges according to their weights to the vertex (i.e. OCR abandons line 1, and lines 8 - 10 of Algorithm: CR () in figure 7).

The method allows not only to improve efficiency (although worst case complexity is the same) but also to process only the best 'ripple' (i.e. heaviest adjacent edge) for each vertex each time. The pseudocode of Ordered Concurrent Rippling algorithm is given in figure 8. The complexity of the algorithm is O (N^2logN) to sort the N-1 neighbors of the N vertices. The algorithm then iterates at most N^2 times. Hence the overall worst case complexity of the algorithm is O (N^2logN).

For each vertex v, v.neighbor is the list of v's adjacent vertices sorted by their similarity to v from highest to lowest.
If v is a centroid (v.centroid == 1); v.cluster contains the list of vertices $\neq v$ assigned to v

Algorithm: CR ( )
1. Sort E in order of the edge weights
2. CentroidChange = true
3. index = 0
4. While (CentroidChange && index < N-1 && E is not empty)
    5. CentroidChange = false
    6. For each vertex v, take its edge e_{vw} connecting v to its next closest neighbor w; i.e. $w = v$.neighbor [index]
    7. Store these edges in S
    8. Find the lowest edge weight in S, say *low*, and empty S
    9. Take all edges from E whose weight >= *low*
    10. Store these edges in S
    11. **PropagateRipple** (S)
    12. index ++
13. For all $i \in V$, if i is a centroid, return i and i.cluster

Sub Procedure: PropagateRipple (list S)
/* This sub procedure is to propagate ripples for all the centroids. If the ripple of one centroid touches another, the heavier weight centroid will engulf the lighter centroid and its cluster. If the ripple of a centroid touches a non-centroid, the non-centroid is assigned to the centroid. A non-centroid can be assigned to more than one centroid, allowing overlapping between clusters, a generally desirable feature */

```
While (S is not empty)
    Take the next heaviest edge, say e_vw, from S
    If v ∉ x.cluster for all x ε V
            If w is a centroid, compare v's weight to w's weight
                If (w.weight > v.weight)
                 add v and v.cluster into w.cluster
                    Empty v.cluster
                    If v is a centroid
                        v.centroid = 0
                        CentroidChange = true
                Else
                 add w and w.cluster into v.cluster
                    Empty w.cluster
                    w.centroid = 0
                    CentroidChange = true
            Else if w is not a centroid
                v.cluster.add (w)
                If v is not a centroid
                    v.centroid = 1
                    CentroidChange = true
```

Fig. 7. Concurrent Rippling Algorithm

For each vertex v, v.neighbor is the list of v's adjacent vertices sorted by their similarity to v from highest to lowest. If v is a centroid (i.e. v.centroid == 1); v.cluster contains the list of vertices $\neq v$ assigned to v

Algorithm: OCR ( )
public CentroidChange = true
index = 0
While (CentroidChange && index < N-1)
    CentroidChange = false
    For each vertex v, take its edge e_{vw} connecting v to its next closest neighbor w; i.e. $w = v$.neighbor [index]
    Store these edges in S
    PropagateRipple (S)
    index ++
For all $i \varepsilon$ V, if i is a centroid, return i and i.cluster

Fig. 8. Ordered Concurrent Rippling Algorithm

3.1.2.3.3 Maximizing Intra-cluster Similarity

The key point of Ricochet is that at each step it tries to maximize the intra-cluster similarity: the average similarity between vertices and the centroid in the cluster.

Sequential Rippling (SR and BSR) try to maximize the average similarity between vertices and their centroids by (1) selecting centroids in order of their weights and (2) iteratively reassigning vertices to the nearest centroids. As in [13], selecting centroids in order of weights is a fast approximation to maximizing the expected intra-cluster similarity. As in K-means, iteratively reassigning vertices to nearest (most similar) centroids decreases the distance (thus maximizing similarity) between vertices and their centroids.

Concurrent Rippling (CR and OCR) try to maximize the average similarity between vertices and their centroids by (1) processing adjacent edges for each vertex in order of their weights from highest to lowest and (2) choosing the bigger weight vertex as a centroid whenever two centroids are adjacent to one another. (1), as in the minimum spanning tree algorithm, ensures that at each step, the best possible merger for each vertex v is found (i.e. after merging a vertex to v with similarity s, we can be sure that we have already found and merged all vertices whose similarity

is better than *s* to *v*); while (2) ensures that the centroid of a cluster is always the point with the biggest weight. Since we define a vertex weight as the average similarity between the vertex and its adjacent vertices; choosing a centroid with bigger weight is an approximation to maximizing the average similarity between the centroid and its vertices.

Further, although OCR is, like its family, a partitional graph-based clustering algorithm; its decision to only process the heaviest adjacent edge (hence the best possible merger) of each vertex at each step, like in the minimum spanning tree algorithm, is compatible to the steps of agglomerative single-link hierarchical clustering. We believe therefore that OCR combines concepts from both partitional and agglomerative approaches. As in [27], such combination has been experimentally found to be successful than either of the methods alone.

Section 3.2 Applications

Here we present our proposed methods on applications. We propose to illustrate and evaluate the performance of our graph-based clustering algorithms for the task of document clustering. We propose to use our MST-Sim clustering algorithm to solve k-member clustering problem which has the constraint that each cluster must have at least k members. We propose to use hierarchical clustering for part-of-speech tagging of Bahasa Indonesia. The use of hierarchical clustering allows for user interactivity in between hierarchical levels of clusters, and the addition of morphological/words constraints in between levels. We propose a novel method using random walk to rank items based on opinions written in their reviews. We also propose a novel method that combines random walk algorithm and Machine Translation technique to compute time series correlation, which allows for time shifting and partial matching between time series.

3.2.1 Document Clustering

We illustrate and evaluate the performance of our graph-based clustering algorithms for the task of off-line and on-line document clustering. We first construct a graph to represent the

documents and their similarity relationships. Documents are vertices of the graph and edges are weighted with the inverse of the tf-idf cosine-similarity of the documents they connect. The graph is a clique. We apply our graph-based clustering algorithms to cluster the vertices in the graphs.

3.2.2 k-member Clustering

Our family of MST-Sim clustering is easily adaptable to k-member clustering problem. We only need to add the new constraint for merging of clusters that two clusters can be merged only if the resulting cluster has size $\leq$ k. When all vertices have been assigned to clusters, every remaining cluster whose size is less than k is merged with another cluster to which it is connected with the lightest edge.

3.2.3 Hierarchical Clustering for POS Tagging of Bahasa Indonesia [56]

We apply hierarchical clustering algorithms to the problem of part-of-speech tagging [56]. POS tagging is the task of assigning the correct class (part-of-speech, word class or lexical category, loosely speaking) to each word in a sentence.

Classes are defined and recognized by means of structural (morphological and syntactical) and semantic criteria. Classes and criteria, while relatively well understood for most Western languages whose grammarians have been busy studying and passionately debating them since classical antiquity, remain an important and fundamental topic of research for other less studied languages. This is particularly the case for languages of the Austronesian family like the dynamic Malay and Indonesian languages with dialects and usages that arguably fall into the group Flexible-Syntactic-Category languages [57].

Clustering is the task of grouping objects according to their features so that objects within the same cluster are similar and objects belonging to different clusters are dissimilar. Clustering determines intrinsic classes in a set of unlabeled objects provided relevant features and metrics.

We explore the design of a tool for the interactive exploration of part-of-speech classes using structural features. At the heart of the tool are incremental hierarchical clustering algorithms. The algorithms are used to detect classes using structural features such as morphological and syntactical ones. The algorithms have been modified or designed to allow interactive exploration (doing and undoing clusters) and constrained clustering (preventing or forcing objects and clusters to merge by means of constraints).

3.2.3.1 Methods

We group words into parts-of-speech classes based on the cosine similarity of their feature vectors. The problem is one of clustering in a dense graph whose vertices are words and edges are weighted by similarity.

We evaluate two hierarchical clustering algorithms for this purpose. Hierarchical clustering is chosen because it does not need the number of clusters to be provided a priori and because the resulting hierarchy of clusters provides a chance for user interactivity in-between the hierarchy levels. The two hierarchical clustering algorithms evaluated are single-link agglomerative hierarchical clustering [7] and our own Borůvka hierarchical clustering. Here we investigate single-link hierarchical clustering because it is closest to the algorithm for finding MST that we exploit in our own Borůvka hierarchical clustering. However, the method can be extended to other hierarchical clustering (average- and complete-link).

3.2.3.1.1 Single-link Agglomerative Hierarchical Clustering

Single-link agglomerative hierarchical clustering treats each vertex as a separate cluster initially. It then scans through the list of edges (from heaviest to lightest), and iteratively merges pairs of clusters connected by the heaviest edge until there is only one cluster left. Single-link agglomerative clustering is essentially Kruskal's algorithm [16] for finding a MST in the edge-weighted graph. The pseudocode is shown in figure 9.

Algorithm: Single-link Agglomerative Hierarchical Clustering ( )

Let *E* be the set of edges in the graph

- Treat each vertex as a singleton cluster at level 0
- Sort *E* from heaviest to lightest edge weights
- While the highest level cluster does not contain all vertices, take the next heaviest edge *e* from *E* (say *e* connects vertices *A* and *B*)
  - If *A* belongs to a non-singleton cluster and *B* is a singleton cluster, include *B* in *A*'s cluster. Similarly, If *B* belongs to a non-singleton cluster and *A* is a singleton cluster, include *A* in *B*'s cluster
  - Else if *A* and *B* belong to different clusters, create a new cluster at the level above the maximum level of the two clusters. Save the information of the two clusters (i.e. their levels, their members) and update their members to belong to the new cluster
- Output the clusters at each level

Fig. 9. Pseudocode for Single-link Agglomerative Hierarchical Clustering

3.2.3.1.2 Borůvka Hierarchical Clustering

Since hierarchical clustering is essentially finding a MST in the edge-weighted graph, we propose our own hierarchical clustering that is based on Borůvka algorithm [17] for finding MST in an edge-weighted graph. Borůvka algorithm treats each vertex as a separate cluster. It then scans through the list of clusters, merging each cluster to another cluster to which it is connected with its heaviest edge, until there is only one cluster left. Borůvka scans through the list of vertices while single-link scans through the list of edges. The pseudocode is shown in figure 10.

Algorithm: Borůvka Hierarchical Clustering ( )

- L = 0
- Treat each vertex as a singleton cluster at level L
- While level L contains more than one cluster
  - While there are still clusters at level L, take a cluster, say C, from this level
    - Find the lightest edge connecting C to another cluster, say D
    - If D is at level L, create a new cluster at level L+1. Save the information of C and D (i.e. their levels, their members); remove C and D from the list of clusters at level L, and update their members to belong to the new cluster
    - Else if D is a cluster at level higher than L, save the information of C (i.e. its level, its members); remove C from the list of clusters at level L, and update C's members to belong to D
  - L++
- Output the clusters at each level

Fig. 10. Pseudocode for Borůvka Hierarchical Clustering

3.2.3.2 A Tool for Interactive POS Exploration

3.2.3.2.1 Feature Vectors

Using the extended Schutze's feature vectors approach [39], we measure similarity of words by the degree to which they share the same two neighbors on the left and on the right, respectively.

The counts of neighbors are assembled into a vector, with one dimension for each neighbor. Each word is represented by four feature vectors: left vector (corresponding to the word's immediate left neighbor), right vector (corresponding to the word's immediate right neighbor), secondary left vector (corresponding to neighbor that precedes the word's immediate left neighbor), secondary right vector (corresponding to neighbor that follows the word's immediate right neighbor). For example, if w1, w2, w3, w4 are neighbors to be considered, the word w – assuming it only occurs in the phrase (w1 w2 w w3 w4) – is represented by its left vector: (0 1 0 0), its right vector: (0 0 1 0), its secondary left vector: (1 0 0 0), and its secondary right vector: (0 0 0 1). In our approach, each feature vector consists of 3000 entries, corresponding to the 3000 most frequent words in the corpus. Each word is therefore represented by a vector of 12000 entries. Similarity between words is measured as cosine of their representative vectors.

Since Indonesian language is rich in derivational morphology that often indicates parts-of-speech, in future we can also incorporate morphological features into the vector. For example, having an entry in the vector that is set to 1 if the word has a certain affix (e.g. "pe-" which often indicates a noun) and is set to 0 if otherwise.

3.2.3.2.2 Interactive Clustering

Borůvka hierarchical clustering forms clusters by levels (cf. Fig. 10). Because of this property, Borůvka hierarchical clustering algorithm can incorporate users' interactivity in-between levels quite easily. After processing each level, resulting clusters can be displayed and user can be asked to input his constraints: which clusters to be broken and which clusters to be merged. The

clusters are then refreshed (broken or merged) to satisfy the constraints and the process repeats. After the user agrees to the clustering at that level, the clusters at the next level can be formed and displayed.

3.2.3.2.3 Constrained Clustering

At each level of the hierarchy of clusters, the clusters are displayed and user can be asked to input his constraints. The constraints can be in the form of words or morphological constraints.

Words constraints dictate which words to exclude from one another (exclusion list) and which words to include to one another (inclusion list). Morphological constraints can be added so that words with the same affixes are grouped in the same cluster. For example words with affix "me-" that often indicates verbs, must be included with words with affix "di-" that often also indicates verbs. The clusters are then refreshed to reflect the constraints and the process repeats.

3.2.4 Opinion-based Ranking

The success of Web 2.0 can be sized by the increasing popularity of forums and media in which users and organizations express and share views on anything and everything. Focused reviews are available on the World Wide Web for items as varied as consumer electronics hotels, public schools and election candidates. Name it; folks have reviewed it!

Unfortunately, the abundance of reviews also makes it challenging for users to compare and rank different items. Although quantitative ratings are often given with textual reviews, these ratings differ in scale, in notation, in criteria, etc. from website to website and hence are hard to aggregate. Some reviews are not even rated. In addition, quantitative ratings have been found to be insufficient in reflecting the opinions of the corresponding textual reviews [58, 59]. The ability to automatically rank different items based on their textual reviews will definitely be beneficial. In our research, we investigate the idea of random walk using PageRank [11] to rank *movies* according to the opinions expressed in their textual reviews.

PageRank has been designed to rank Web pages. Unlike hypertext documents where edges are explicitly available (hyperlinks), there are no obvious edges that can be derived from movie reviews to build a graph where movies are vertices. Comparing movies based on general textual similarities of their reviews may not be entirely appropriate as movies tell different stories. The general textual similarities also do not reflect differences in the opinions expressed about the movie. We could then consider looking for sentences like "movie A is better than movie B", which make direct comparison between movies. Yet, such sentences are rarely found in individual reviews. Instead, in the reviews we commonly find sentences such as "I had a great time" and "the movie was horrible" which are expressing an opinion about the movie by means of adjectives. "Great" suggests a positive opinion. "Horrible" suggests a negative opinion. Whether an adjective expresses a positive or negative opinion is referred to as its semantic orientation. Other researchers have also studied the semantic orientation of adjectives to infer opinion [43, 60, 61]. We also very commonly find sentences such as "this movie is good and funny" or "this movie is boring but has a good ending" in the reviews. The collocation of adjectives in such sentences forms, reinforces and amends the opinion expressed.

Although some adjectives may have some positive or negative universal semantic orientation (e.g. "good", "excellent", "bad", "poor") other adjectives' orientation may not be known or depend on context [44, 62, 63, 64]. The design of effective context-dependent methods is generally considered a challenge in natural language processing [62]. We propose to use PageRank with a graph where vertices are adjectives and edges represent collocation (i.e. context-dependent clues on the semantic relationships between adjectives). Starting from a set of known adjectives (i.e. adjectives that have positive or negative universal semantic orientations), PageRank propagates their semantic orientations to other vertices whose orientations are not yet known and computes the semantic orientation scores. We can then rank movies by computing their individual scores from the semantic orientation scores of the adjectives in the movies'

reviews. The higher the score of its adjectives, the more positive the opinions expressed about a movie: the higher the rank of the movie.

The semantic orientation of an adjective may also depend on further facets of its context. For example, the adjective "funny" may have a positive semantic orientation when used in the review of a comedy movie: "the movie is so funny I had a good laugh", but may have a negative semantic orientation when used in the review of an action movie: "the villain looks a bit funny it was weird". We can therefore build the graph of adjectives for different context and granularity: we can build a single graph based on all reviews, we can build a graph by genre, or a graph by movie (we could also build graph by authors, by date or any other facets of context). In our research we investigate the results for graph built from all reviews, graph built from reviews grouped by genre (e.g. comedy, action, horror etc.) and graph built from reviews grouped by movie. We propose a practical context-dependent ranking procedure that can rank movies directly from their user reviews with no other resource required. The procedure is threefold. We first propose a simple yet effective method for constructing a weighted graph of adjectives from reviews. We use part-of-speech tagging, collocation and pivot words such as conjunctions (e.g. "and") and adverbs (e.g. "however") to create the graph. We refer to this graph as the sentiment graph. The graph is then used to compute semantic orientation scores of individual adjectives using PageRank. The scores of the individual adjectives from all the movie's reviews are combined to get the movie's score. The movies are ranked according to their scores. We detail our proposed method in the next sub sections.

3.2.4.1 Sentiment Graph

The sentiment graph is constructed as follows. We define three variants of the method, depending on whether we construct a sentiment graph from reviews grouped by movie, a sentiment graph from reviews grouped by genre, or a sentiment graph from all reviews. We refer to these variants as individual_, byGenre_, and all_ respectively.

The text of the reviews to be processed is first tagged using a part of speech tagger to identify adjectives. It is also segmented into sentences. We use Brill's part-of-speech tagger [65] and Ratnaparkhi's sentence splitter [66].

We then extract adjectives from the text of the reviews. The adjectives constitute the vertices of the graph. Here, we have assumed that the adjectives in the reviews are related to the movie. There may be other adjectives in the reviews that may not be related to the movie. For example the adjective "terrible" in the sentence "I watch this movie in a terrible cinema". However, we believe that such usage of adjectives (which is not related to the movie in review) is infrequent; therefore its effect can be minimized when we take a large number of reviews.

There exists an edge between two vertices if the corresponding adjectives occur in the same sentence (i.e. if they collocate). The weight of the edge is commensurate to the number of sentences in which the two adjectives collocate. Collocation between adjectives indicates either reinforcement or amendments of semantic orientations between the adjectives.

We obtain a graph G_{pn}=<N, E_{pn}> with N its set of vertices and E_{pn} its set of weighted edges. The weight $W_{pn}(i, j)$ of the edge between the vertices i and j is the number of collocations. W_{pn} is a |N| x |N| matrix called the adjacency matrix. For example, given the sentences "the story is typical yet the ending is surprising", "the movie is typical but the twist at the end is delightful", "I think it is good that the movie has a surprising ending", and "it is typical and has poor plot", G_{pn} is shown in figure 11 (the number in the square brackets indicate the weight of the edge).

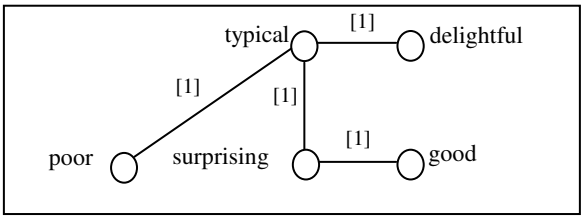

Fig. 11. The graph G_{pn}

If two adjectives occur in a sentence where they are separated by words like “but”, “although” or articulated in simple constructions such as “even if …, …” we refer to this situation as negative collocation. Negative collocation between adjectives indicates amendments of semantic orientations between the adjectives.

We obtain a graph G_n=<N, E_n> with N its set of vertices and E_n its set of weighted edges. The weight $W_n(i, j)$ of the edge between the vertices i and j is the number of negative collocations. W_n is G_n’s adjacency matrix. For example, given the sentences “the story is typical yet the ending is surprising”, “the movie is typical but the twist at the end is delightful”, “I think it is good that the movie has a surprising ending”, and “it is typical and has poor plot”, G_n is shown in figure 12 (the number in the square brackets indicate the weight of the edge).

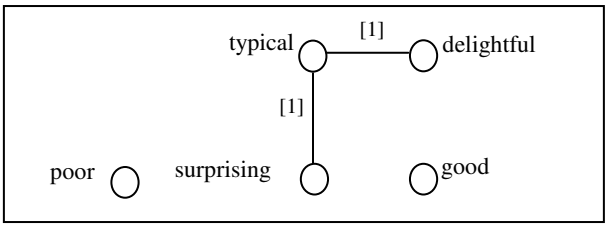

Fig. 12. The graph G_n

If two adjectives are negatively collocated to the same adjective in different sentences, we treat them as being positively collocated. For example, if we have two sentences: “the story is typical yet the ending is surprising” and “the movie is typical but the twist at the end is delightful” the adjectives “surprising” and “delightful” are considered positively collocated. We compute the co-citation matrix W_c of W_n [67]. Positive collocation between adjectives indicates reinforcement of semantic orientations between the adjectives.

The final sentiment graph G is a structure <N, E> with N its set of vertices and E its set of weighted edges where

$$W(i, j) = (W_{pn}(i, j) - W_n(i, j) + W_c(i, j)) / \sum (W_{pn}(k, j) - W_n(i, j) + W_c(k, j)) \qquad (14)$$

for k = 1 to N. W is the adjacency matrix of our sentiment graph. For example, given the sentences “the story is typical yet the ending is surprising”, “the movie is typical but the twist at the end is delightful”, “I think it is good that the movie has a surprising ending”, and “it is typical and has poor plot”, G is shown in figure 13 (the number in square brackets indicates edge weight).

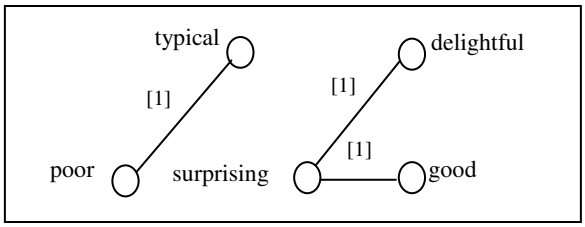

Fig. 13. The graph G

3.2.4.2 PageRank

PageRank algorithm is applied to the sentiment graph obtained in the previous section to compute the semantic orientation scores of adjectives.

We define two sets containing known adjectives with positive and negative semantic orientation respectively. We assign non-zero initial semantic orientation scores to these adjectives. These semantic orientation scores will be propagated to other adjectives during the course of PageRank application on the graph. In this manner, the semantic orientation scores of unknown adjectives can be computed. The set Good_Adjectives is the set containing known adjectives with positive orientation. The set Bad_Adjectives is the set containing known adjectives with negative orientation.

In our method, we define the set Good_Adjectives to contain 19 adjectives: “good” and its synonyms: “excellent”, “brilliant”, “well”, “better”, “best”, “worthy”, “worth”, “nice”, “great”, “perfect”, “positive”, and negative antonyms: “not bad”, “not horrible”, “not terrible”, “not awful”, “not worse”, “not worst”, “not negative”. We define the set Bad_Adjectives to be the exact mirror image of Good_Adjectives (i.e. it contains 19 adjectives: “not good”, “not

excellent", "not brilliant", etc.). Other choice of adjectives for Good_Adjectives and Bad_Adjectives is certainly possible. We investigate the sensitivity of our method to the choice of adjectives defined in Good_Adjectives and Bad_Adjectives in our experiments.

The vertex in the graph is assigned a non-zero initial semantic orientation score if the corresponding adjective is in the set Good_Adjectives or Bad_Adjectives, and is assigned zero initial semantic orientation score otherwise: i.e. we construct a vector $a_p^0 = <a_1 \ldots a_{|N|}>$ in which a_i is $1/|\text{Good\_Adjectives} \cap N|$ if the corresponding adjective is in Good_Adjectives and 0 otherwise, and we construct a vector $a_n^0 = <a_1 \ldots a_{|N|}>$ in which a_i is $1/|\text{Bad\_Adjectives} \cap N|$ if the corresponding adjective is in Bad_Adjectives and 0 otherwise.

PageRank [11] computes the fix point or stable state of the product of an adjacency matrix with itself and a damping factor (the probability, at any step, that a walker will continue walking). This is similar to a random walk in the graph defined by the matrix, for a walker getting fatigued and switches to a random vertex according to the damping factor.

The input to PageRank algorithm is the adjacency matrix W and a vector a^0 (a_p^0 or a_n^0), which is the initial semantic orientation scores assigned to the vertices (adjectives).

In the formula below, α is the damping factor and e represents the probability that a random walker will choose a random vertex when it gets tired. As in [11] this probability is set to be equal for all the vertices. PageRank algorithm iteratively computes the semantic orientation scores of the vertices (adjectives), i.e. the vector a:

$$a^k = \alpha W a^{k-1} + (1- \alpha) e \qquad (15)$$

As in [11], we set α to be 0.85. $e = <e_1 \ldots e_{|N|}>$ is constant across iterations. We set $e_i = 1/|N|$ for any i as in [11].

When we use a_p^0, we propagate the semantic orientation scores of known positive adjectives to other adjectives in the graph. Correspondingly, when we use a_n^0, we propagate the semantic

orientation scores of known negative adjectives to other adjectives in the graph. Therefore, depending whether we use a_p^0 or a_n^0, we obtain methods that compute positive or negative semantic orientation scores of the adjectives in the graph, respectively. We refer to these methods as _Positive and _Negative, respectively.

The vertices (adjectives) can also be ranked according to their semantic orientation scores to produce context-dependent ranking of adjectives.

The positive (resp. negative) score of each movie is computed as the sum of positive (resp. negative) scores of adjectives from all its reviews. The sum considers duplicates, i.e. an adjective that appears twice in the reviews will contribute its score twice towards the total sum. In our method, we use summation to combine the scores of the adjectives. Other function is certainly possible and can be explored in future.

Depending on how we construct a sentiment graph, we define three variants: (1) individual_: we construct a sentiment graph from reviews grouped by individual movie, (2) byGenre_: we construct a sentiment graph from reviews grouped by genre, and (3) all_: we construct a sentiment graph from all reviews.

Depending on our input to PageRank, we define two variants: (1) _Positive: we input the matrix W and the vector a_p^0 to PageRank to compute positive semantic orientation scores of adjectives, (2) _Negative: we input the matrix W and the vector a_n^0 to PageRank to compute negative semantic orientation scores of adjectives.

Using the computed positive and negative semantic orientation scores, we define another variant called _PositiveNegative which computes positive semantic orientation score minus negative semantic orientation score.

Therefore in total we propose 9 methods: (1) individualPositive, (2) individualNegative, (3) individualPositiveNegative, (4) byGenrePositive, (5) byGenreNegative, (6) byGenrePositiveNegative, (7) allPositive, (8) allNegative, and (9) allPositiveNegative.

3.2.5 Time Series Correlation [68]

We propose to measure correlation between time series that allows for time-shifting and partial matching. We present a new definition of correlation between two time series by representing the time series using their gradient sequences. The gradients serve as the vessel of change: two time series are correlated if the changes (gradient) in one are related to the changes in the other. Thus, the correlation measure is reduced to that of measuring similarity between the gradient sequences. We contribute a novel method of measuring similarity that allows for time shifting and partial matching of the gradient sequences. Inspired by ideas from machine translation, we use assignment of conditional probabilities to find the maximum alignment between two sequences that allows for some mismatching. Our maximum fuzzy alignment method is also able to exhaustively enumerate all possible alignments between the sequences by transforming the problem to that of matrix multiplication to convergence.

In this research, we measure only positive correlation. We define two time series to be positively correlated if they have high similarity in their corresponding gradient sequences. In practical applications, we are often more interested in positive correlation: e.g. whether the success of one company causes the success of its partner company. Negative correlation is harder to deduce because the success of one company may not always cause the failure of its competitor.

Our motivation is to find a matching sub-sequence between two gradient sequences that is longest, allows for some shifting and mismatch, and can start and stop from and to any position within the sequences: i.e. allow for partial similarity. Our idea is based on techniques from machine learning, where the best alignment between words in two sentences can be found using

conditional probability. For example, given two sentences (one in English ($E_{sentence}$) and another in Indonesian ($I_{sentence}$)):

E$_{sentence}$: I want to eat good food

I$_{sentence}$: Saya ingin makan makanan yang enak

Each word in $E_{sentence}$ can be mapped to a word in $I_{sentence}$. Each word has many possible mappings. For example, possible mappings for the word "want" are: (want, Saya), (want, ingin), (want, makan), (want, makanan), etc.

Further, there are many possible alignments (i.e. sequence of mappings) between two sentences. For example, possible alignments between $E_{sentence}$ and $I_{sentence}$ are: {(I, ingin), (want, makan), (to, makanan), (eat, makan), (good, yang), (food, enak)}, {(I, Saya), (want, ingin), (to, makan), (eat, makanan), (good, enak), (food, yang)}, etc. Conditional probability is used to find the best possible sequence of mappings which in this case is {(I, Saya), (want, ingin), (to, null), (eat, makan), (food, makanan), (good, enak)}.

Let m_i be the i^{th} mapping in the sequence, and m_i maps the word E_{mi} in $E_{sentence}$ to the word I_{mi} in $I_{sentence}$; i.e. $m_i = (E_{mi}, I_{mi})$. The best possible alignment is the sequence of mappings that maximizes the probability:

$$P(m_1 \dots m_n) = \Pi P(m_i | m_{i-1}) * P(E_{mi}, I_{mi}) \qquad (16)$$

Where $1 < i < n$; $m_i = (E_{mi}, I_{mi})$; and $P(E_{mi}, I_{mi})$ is the probability that the word E_{mi} is mapped to the word I_{mi}.

We translate this idea to find the best matching sub-sequence between two gradient sequences. To implement our method more efficiently, we first utilize the technique of PAA [69] to segment the gradient sequence and then compress the gradient sequence into a character sequence. The use of probability instead of exact matching will allow for some mismatch or fuzzy matching

between the two character sequences. Now, each mapping is between two characters in the sequences, and the task is to find the best alignment: i.e. the sequence of these mappings ā:

$$\bar{a} = \text{argmax}_a \text{ (Score (a))}; \text{ where } a = m_1, m_2 \ldots m_n \qquad (17)$$

$$\text{Score (a)} = \sum_{i=1}^{n} P(m_i \mid m_{i-1}) * K(m_i) \qquad (18)$$

$$\text{Score (a)} = K(m_1) + \sum_{i=2}^{n} P(m_i \mid m_{i-1}) * K(m_i) \qquad (19)$$

Where $m_i = (E_{mi}, I_{mi})$ and K (m_i) is the kernel function that measures the similarity between the values of the two characters E_{mi} and I_{mi} in the sequences. We treat the probability of choosing m_1 as the first mapping, i.e. P (m_1) to be uniform for all possible m_1.

Consider two character sequences F = f_1, f_2 … f_n and T = t_1, $t_2 \ldots t_n$. Since time can only move forward, the mapping (f_1, t_1) can only be followed by the mapping (f_2, t_y) or (f_y, t_2) where y > 1. Using this idea, we define:

$$P(m_i \mid m_{i-1}) = \begin{cases} P(t_{mi} \mid t_{mi-1}) \text{ if } mi=(f_{x+1}, t_{mi}) \text{ and } m_{i-1}=(f_x, t_{mi-1}) \\ P(f_{mi} \mid f_{mi-1}) \text{if } mi=(f_{mi}, t_{x+1}) \text{ and } m_{i-1}=(f_{mi-1}, t_x) \\ 0 \text{ otherwise} \end{cases} \qquad (20)$$

We can define P $(t_{mi} \mid t_{mi-1})$ to be uniform for any t_{mi}, t_{mi-1}; or we can define it such that if t_{mi} is closer to t_{m-1} (closer in terms of distance between the two characters' positions in the sequence) we give it higher probability. This will result in an alignment that prefers mappings of nearby characters in the sequences. For the purpose of our experiment, we use this second option of weighing probability based on distance. The same probability assignment is used for P $(f_{mi} \mid f_{mi-1})$.

Note that our method also allows us to specify the limit on the number of allowed mismatch (say *d*) by assigning non-zero probabilities P $(t_{mi} \mid t_{mi-1})$ (and P $(f_{mi} \mid f_{mi-1})$ respectively) for t_{mi} and t_{mi-1} (and f_{mi} and f_{mi-1} respectively) which are at most *d* distance away from each other in the sequence and assigning 0 otherwise.

Since we would like to consider all possible alignments of the sequences (all alignment (partial/full) that can start and stop from and to any mapping of any characters in the sequences), potentially we have $2^{|F|+|T|}$ comparisons to do because $2^{|F|}$ subsequences from the first sequence can be mapped to $2^{|T|}$ subsequences from the second sequence. To reduce this exponential complexity, we formulate the computation of all possible alignments a, and their scores using a *modified* matrix multiplication, based on random walks on the graph represented by the matrix p^r, until convergence:

$$p^{r+1}(i, j) = \max (p^r(i, k) * p^0(k, j)) \qquad (21)$$

Each entry (i, j) in the |FT| x |FT| matrix p^r will represent the maximum probability of an alignment starting from the mapping m_i to the mapping m_j via r other mappings. We initialize $p^0(i, j) = P(m_j \mid m_i)$. The matrix multiplication will converge in at most max (|F|, |T|) because there are only so many characters to map in between m_i and m_j. Notice also that the matrix is sparse because from each mapping you can only move to a limited number of other mappings (since we have assumed that time can only move forward). The non-zero entries in the matrix is at most c * |F| * |T| where $c \leq (2 * \max (|F|, |T|))$.

We call our matrix multiplication 'modified' because when we multiply each entry in row i to each entry in column j; we do not sum them up instead we pick the maximum.

Since we are interested not only in the probability but also the scores of the alignment (that takes into account the kernel function between the values of the mapped characters), we maintain another matrix M to keep these scores. Each entry (i, j) in the matrix M^r represents the maximum scores of the alignment starting from the mapping m_i to the mapping m_j via r other mappings:

$$M^{r+1}(i, j) = \max (M^r(i, k) + M^r(k, j)) \qquad (22)$$

We initialize $M^0(i, j) = P(m_j \mid m_i) * K(m_j)$. Once the matrix p^r converges, we pick the alignment with the maximum scores as the entry (i, j) in M^r that maximizes $K(m_i) + M^r(i, j)$, say this entry

is M^r (i_{max}, j_{max}). The alignment from the mapping i_{max} to the mapping j_{max} will be the alignment with the highest score.

CHAPTER 4. EXPERIMENTAL SETUP

Here we present our experimental setup on how we illustrate and evaluate the performance of our proposed methods.

Section 4.1 Graphs

4.1.1 Graph Algorithms

4.1.1.1 Random Walk

We illustrate and evaluate the performance of our proposed methods which are based on random walk for the task of opinion mining – to rank movies based on opinions in their reviews, and for the task of computing time series correlation – to compute correlation between time series in both synthetic and real data sets. Experimental setup for these two applications are detailed in the applications section 4.2.4 and 4.2.5 respectively.

4.1.1.2 A Variant of Randomized MST Algorithm

We illustrate and evaluate the performance of our variant of randomized MST algorithm by applying the algorithm to find the MST of a complete graph: a bi-directed graph in which there are no self loops, no more than one edge between any two distinct vertices, and where every pair of distinct vertices is connected by an edge. We construct the graph randomly – each edge in the graph is assigned a random weight between 0 and 1.

Because our variant of randomized MST Algorithm is exact, it produces a minimum spanning tree that is similar to that produced by other MST algorithms. Hence we measure instead the efficiency of our proposed algorithm, in terms of the run time of our algorithm (in milliseconds), when it is applied to synthetic graphs with varying number of vertices and edges.

4.1.2 Graph-based Clustering Algorithms

We illustrate and evaluate the performance of our graph-based clustering algorithms for the task of off-line and on-line document clustering. The experimental setup for document clustering can be found in section 4.2.1 on document clustering.

Section 4.2 Applications

4.2.1 Document Clustering

We illustrate and evaluate the performance of our graph-based clustering algorithms for the off-line and on-line document clustering task. Documents are vertices of the graph and edges are weighted with the inverse of the tf-idf cosine-similarity of the documents they connect. The graph is a clique. We evaluate the performance of our proposed algorithms empirically against other state of the arts clustering algorithms: Markov Clustering, Star Clustering, K-means, Single-link Hierarchical Clustering, and Zahn's MST Clustering. For Star clustering, by default and unless otherwise specified, we set the value of threshold σ for Star clustering to be the average similarity of documents in the given sub-collection.

We use data from Reuters-21578 [70], TIPSTER–AP [71] and our original collection: Google. The Reuters-21578 collection contains 21,578 documents that appeared in Reuter's newswire in 1987. The documents are partitioned into 22 sub-collections, each of the first 21 sub-collections contain 1000 documents while the last sub-collection contains 578 documents. For each sub-collection, we cluster only documents that have at least one explicit topic (i.e. document that has at least one topic category within its <TOPICS> tag).

The TIPSTER–AP collection contains AP newswire from the TIPSTER collection. For the purpose of our experiments, we have partitioned Tipster-AP into 2 separate sub-collections with 1787 and 1721 number of documents respectively.

Our original collection: Google contains 2029 news documents obtained from the Google News website in December 2006 [72]. The Google documents have been labeled manually and partitioned into 2 separate sub-collections. In total we have 26 sub-collections. The sub-collections, their number of documents and topics/clusters are reported in Table 1.

We study effectiveness (recall, r, precision, p, and F1 measure, F1 = (2 * p * r) / (p + r)), and efficiency in terms of running time. In each experiment, for each topic, we return the cluster which best approximates the topic. Each topic is mapped to the cluster that produces the maximum F1-measure with respect to the topic:

$$\text{topic}(i) = \text{argmax}_j \{F1(i, j)\} \qquad (23)$$

where F1 (i, j) is the F1 measure of the cluster number j with respect to the topic number i. The weighted average of F1 measure for each sub-collection is calculated as follows:

$$F1 = \Sigma (t_i/S) * F1(i, \text{topic}(i)); \text{ for } 0 \leq i \leq T \qquad (24)$$

$$S = \Sigma t_i; \text{ for } 0 \leq i \leq T \qquad (25)$$

where T is the number of topics in the sub-collection; t_i is the number of documents belonging to topic i in the given collection. For each sub-collection, we calculate the weighted-average of precision, recall and F1-measure produced by each algorithm. We then present the average results produced by each algorithm over each collection. Our estimation of global performance is therefore macro-averaging.

Whenever we perform efficiency comparison with other graph-based clustering algorithms (Markov, Star, Zahn, Single-link), we do not include the pre-processing time to construct the graph and compute all pair wise cosine-similarities. The pre-processing time of these algorithms is the same as our proposed algorithms which are also graph-based. Furthermore, the potentially large complexity of pre-processing, O (n^2), may undermine the actual running time of the algorithms themselves. We however include this pre-processing time when comparing with non

graph-based clustering algorithms such as K-means that do not require graph or all pair wise similarities to be computed. This is to illustrate the effect of pre-processing on the efficiency of graph-based clustering algorithms.

Table 1 Description of Collections

Sub-collection	**# of docs**	**# of topic**	**Sub-collection**	**# of docs**	**# of topic**
reut2-000.sgm	981	48	Reut2-001.sgm	990	41
reut2-002.sgm	991	38	Reut2-003.sgm	995	46
reut2-004.sgm	990	42	Reut2-005.sgm	997	50
reut2-006.sgm	990	38	Reut2-007.sgm	988	44
reut2-008.sgm	991	42	Reut2-009.sgm	495	24
reut2-010.sgm	989	39	Reut2-011.sgm	987	42
reut2-012.sgm	987	50	Reut2-013.sgm	658	35
reut2-014.sgm	693	34	Reut2-015.sgm	992	45
reut2-016.sgm	488	34	Reut2-017.sgm	994	61
reut2-018.sgm	994	50	Reut2-019.sgm	398	24
reut2-020.sgm	988	28	Reut2-021.sgm	573	24
Tipster-AP1	1787	47	Tipster-AP2	1721	48
Google1	1019	15	Google2	1010	14

4.2.2 k-member Clustering

One of the premises of our work is the opportunity to easily adapt our MST-Sim clustering algorithms to related problems of local and dynamic nature such as k-member clustering. We illustrate and evaluate the performance of our MST-Sim algorithm for k-member clustering with the k-member greedy clustering algorithm proposed in [32], which has been experimentally shown to outperform Median Partitioning and K-Nearest Neighbor algorithms for k-

anonymization in terms of information loss. Records as vertices of the graph and edges are weighted with distance metric in [32].

We use the Adult dataset from [73], which is considered a benchmark for evaluating the performance of k-anonymity algorithms. The data set is prepared as described in [32, 74]. We remove records with missing values, resulting in 30,162 records, and use eight of the original attributes: *age*, *work class*, *education*, *marital status*, *occupation*, *race*, *gender*, and *native country* as quasi-identifiers. *Age* and *education* are numerical attributes while the other six attributes are categorical attributes using the assumptions, mappings and taxonomies of [32, 74] and a taxonomy of world regions from [75]. In details, for *work class* we use the taxonomy from [74]. For *education*, we use the mappings to numeric value from [74]. For *occupation*, *race*, and *gender*, we use the flat taxonomy mentioned in [32]. For *native country*, we use the taxonomy of world regions from [75]. Since the taxonomy tree for *marital status* is not presented in [32, 74], we use the taxonomy tree (cf. figure 14) that has the same height as that used in [74, 76].

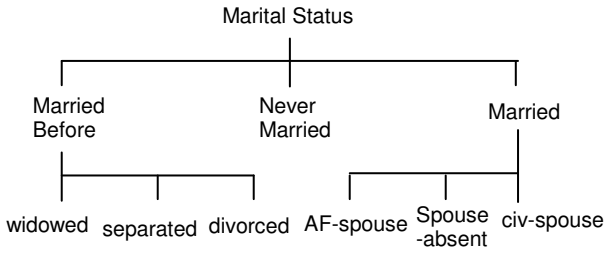

Fig. 14. Taxonomy Tree for Marital Status

We run our proposed algorithms and k-member greedy clustering algorithm all on the same data and present the results. We present the effectiveness (in terms of total information loss [32] and standard deviation) and efficiency comparison. The only difference between the computation of information loss and standard deviation is in the computation for numerical attributes. Information loss of a numerical attribute *A* in a cluster *C* is measured as the range of values of *A*

in *C* divided by the maximum range of values of *A* in the whole dataset [32]. On the other hand, standard deviation of *A* in *C* measures of how much the values of *A* deviate from its mean in *C*. Standard deviation is a measure of tightness of a cluster.

Information loss and standard deviation for categorical attributes are measured in the same manner as in [32]: given a categorical attribute *B*, its information loss/standard deviation in a cluster *C* is the height of the lowest common ancestor of the values of *B* in *C* (as found in the taxonomy tree of attribute *B*) divided by the height of the tree.

4.2.3 Hierarchical Clustering for POS Tagging of Bahasa Indonesia

We illustrate and evaluate the proposed methods using the Indonesian language corpus used in [77]. The corpus is made of 3000 sentences; each sentence is a document from Indonesian online newspaper Kompas, dated from January – June 2002 inclusive.

We obtain 3000 most frequent words in the corpus to compose the feature vectors. Out of these 3000 words, we select 198 words whose POS tags are not ambiguous. Since no pre tagged corpus is available for the Indonesian language, we manually tag these 198 words using tags inspired by Penn Treebank tag set [78]. The words and their tags are listed in the appendix. We study recall, r, precision, p, and F1 measure; F1 = (2 * p * r) / (p + r).

For each hierarchy level and each part-of-speech, we return the cluster which "best" approximates the part-of-speech, i.e. at each level, each part-of-speech *i* is mapped to the cluster *j* which produces maximum F1-measure with respect to the part-of-speech:

$$F1\_POS\ (i) = \text{argmax}_j\ \{F1\ (i, j)\} \qquad (26)$$

where F1 (I, j) is the F1 measure of the cluster number *j* with respect to the parts-of-speech *i*. The weighted average of F1 measure for each hierarchy level is calculated as:

$$WeightedAve\_F1 = \Sigma\ (n_i/S) * F1\ (I, F1\_POS\ (i)) \qquad (27)$$

for $0 \leq i \leq T$; where T is the number of parts-of-speech; n_i is the number of words belonging to parts-of-speech i; and S is the number of words (i.e. S = 198).

4.2.4 Opinion-based Ranking

We illustrate and evaluate our method for ranking movies based on opinions in their reviews, by using reviews of box office movies written by users of a popular movie review site. We pick 50 movies randomly from box office list of November 2007 to February 2008. For each movie, we download all its users' reviews. For each movie we note its box office figure, its overall quantitative user rating, and its genre. The movies are of genre action, animation, children, comedy, drama, foreign film, horror, musical, romance, science fiction, chick flick, crime, political, or psycho.

In our data, we have quantitative user rating for each user and each movie (on a scale of 1 to 10 stars). These ratings are averaged to provide an overall user rating for the movie. However, there are evidences [58, 59] that such rating for measuring reviews is not reliable. The unreliability of user quantitative rating is attributed to its inconsistency [58] and its too coarse granularity [59]. Similar qualitative textual reviews can yield very different quantitative ratings from users. In the most extreme case, the users do not understand the rating system and give a 1 instead of a 10. Choosing a number between 1 and 10 to quantify one's opinion is subjective and difficult [58]. The coarse granularity of the rating scale (1 to 10 stars) for measuring reviews has the underlying assumption that the opinion in textual review is perfectly classified (summarized) into the 10 classes of the star rating [59]. Yet the findings in [59] clearly indicate that the actual text contains significantly more information than the ratings. The loss of information due to the mapping from textual reviews to the coarser star rating is irretrievable. Inconsistency and coarse granularity are the paradox of user rating because to reduce one will mean to increase the other. For example, although it is easier to be consistent when choosing between "good" or "bad instead of choosing a number from 1 to 10; the 2 classes of rating (coarser granularity) loses

more information than the 10 classes of rating (finer granularity). We further investigate the effectiveness of user ratings in our experiments.

In our data, we have also objective figures that represent the opinions of the general audience: i.e. the box office figures. The box office figure is the gross income of the movie, which is the number of tickets sold (indicates the audience's decision to watch the movie) times the price of the ticket (indicates the audience's willingness to pay).

If we assume that reviewers are representative of the general audience and that reviews are representative of the general audience's opinions, then the box office figures should be a suitable source of reference ranking to measure performance in our experiments. We recognize that box office figures may not be the only robust and objective source of reference ranking; it is however an important and valuable one, especially from the marketing point of view.

We construct the sentiment graph and run PageRank on the graph. PageRank computes semantic orientation scores of each vertex (adjective) in the graph. We aggregate the semantic orientation scores of adjectives in the reviews of a movie to determine the semantic orientation score of the movie. We rank the movies according to their scores.

4.2.4.1 Metrics for Measuring Ranking

We present three metrics for measuring ranking performance.

The first metric is *Percentage of Overlap* [79] which is the size of the overlap between two top-k lists: i.e. how many movies in the top-k list of the box office ranking are in the top-k list of our ranking. We normalize this measure by dividing it with k to get the *Percentage of Overlap*. The bigger the overlap, the better is our ranking in matching the box office ranking. k can take a value between 1 to N_{item}, where N_{item} is the total number of movies.

The second metric is *Average Rank Error*. For each movie, we compute the difference between the rank we produce for the movie and the movie's box office rank. *Average Rank Error* is the

average of these rank differences. The smaller the average, the better is our ranking in matching the box office ranking.

The third metric is *Percentage of Rank Overlap* which is the percentage of movies out of the total number of movies that have the same numerical rank in our ranking as in the box office ranking. This is a stricter measure than the *Percentage of Overlap* metric [79] which does not care about the actual numerical ranks. The bigger the rank overlap, the better is our ranking in matching the box office ranking.

4.2.4.2 Methods for Evaluating Ranking

We present two methods for evaluating ranking performance.

We can evaluate the ranking of the entire data or we can evaluate the ranking of just the subset (top-k) of the data. We call this method of evaluating ranking *Top-k*, for k = 1 (we evaluate the ranking of the top 1 movie), k = 2 (we evaluate the ranking of the top 2 movies), to k = 50 (we evaluate the ranking of the entire data).

We can evaluate the ranking of individual movies or we can evaluate the ranking of the groups of movies. We call this method of evaluating ranking *Granularity-g*, for g = 1 (we evaluate the ranking of individual movies), g = 2 (we group movies by 2 based on their scores (i.e. highest 2 movies, second highest 2 movies, third highest 2 movies, etc.), assign each group a rank, and evaluate the ranking of the groups), to g = 50 (we group all movies in one group, assign this group a rank, and evaluate this coarsest ranking).

We introduce coarser granularity ranking (g > 1) because we believe users may often be more interested to know which group of movies is good, which group of movies is medium, and which group of movies is bad instead of the individual ranking of each movie.

In order to measure the effectiveness different granularity of ranking, we use the metric introduced by [32] to measure the coarseness of grouping in terms of information loss. In this

model, information loss for a given movie is proportional to the size of the interval of box office figures of the movies in its group. Total information loss is the sum of information loss of all movies in the data set. We oppose ranking effectiveness to the information loss that a coarser ranking incurs. As far as we know, this is a novel way of measuring ranking.

A combination of the first and second method is possible and can be explored in future. For example, we can consider grouping the movies then evaluating top-k lists of each group or we can consider grouping the movies then evaluating the ranking of the top-k groups only.

4.2.5 Time Series Correlation

We illustrate and evaluate the performance of our proposed method that computes time series correlation as similarity between their corresponding gradient sequences. We use correlation measures produced by our method to identify pairs of time series with high similarity in their gradient sequences. We conduct the evaluation on both synthetic and real time series data.

CHAPTER 5. EXPERIMENTAL RESULTS

Here we present the results of the experiments we have conducted

Section 5.1 Graphs

5.1.1 Graph Algorithms

5.1.1.1 Random Walk

We present the results of experiments to measure the performance of our methods that are based on random walk, in ranking movies based on opinions in their reviews (section 5.2.4) and in computing correlation between time series in synthetic and real data sets (section 5.2.5).

5.1.1.2 A Variant of Randomized MST Algorithm

Here we present the result of experiments to measure the efficiency of our proposed randomized MST algorithm. We compare the efficiency performance of our algorithm with that of the original randomized MST algorithm. We conduct experiments to find the MST of a complete graph: a bi-directed graph in which there are no self loops, no more than one edge between any two distinct vertices, and where every pair of distinct vertices is connected by an edge. Given a graph G = (V, E) where N = |V| and M = |E|, a complete graph with N vertices has M = N (N-1) /2 edges.

We construct 10 complete graphs on which we conduct the experiments. Each graph has different number of vertices: N=100, N=150, N=200, N=250, etc. Each edge in the graph is assigned a random weight between 0 and 1.

The efficiency performance of our algorithm – in terms of its run time (in milliseconds) on graphs with different number of vertices is shown in figure 15. From figure 15, we can see that our algorithm which runs in O (MN) is much slower than the original randomized MST algorithm which runs in Θ (M). Although our algorithm is simpler in implementation, the

original randomized MST algorithm is superior in terms of efficiency due to its usage of Borůvka's steps to recursively reduce the space of the problem.

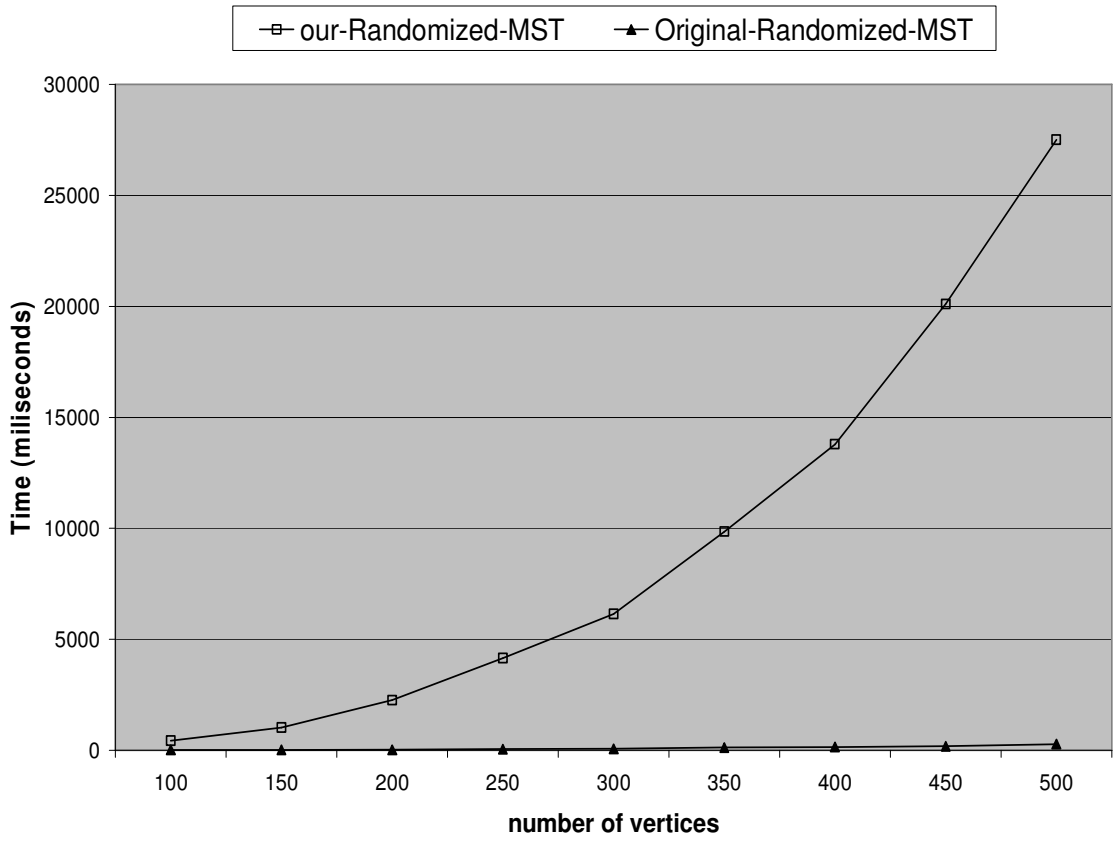

Fig. 15. Efficiency on Complete Graphs

5.1.2 Graph-based Clustering Algorithms

5.1.2.1 Star Clustering

Here we present the comparison of performance of our variants of Star clustering algorithm with the original off-line and on-line Star clustering algorithms and with the restricted and unrestricted extended Star clustering algorithms in the task of off-line and on-line document clustering.

5.1.2.1.1 Performance of Off-line Algorithms

We empirically evaluate the effectiveness (precision, recall, F1 measure) and efficiency (time) of our proposed off-line algorithms: Star-markov, Star-lb, Star-sum, Star-ave and compare their

performance with the original Star algorithm: Star and its variant: Star-random that picks star centers randomly. The results reported for Star-random are average results for various centroids that give us a base line for comparison.

In figure 16 we see that Star-lb and Star-ave achieve the best F1 values on all collections. This is not surprising because as we argued, lb(v) metrics maximizes intra-cluster similarity. The fact that our Star-ave achieves comparable F1 to Star-lb is evidence that average is a sufficient metric for selecting star centers. In figure 16, based on F1: when compared to original Star, our proposed algorithms: Star-ave and Star-markov by far outperform the original Star on all collections. An interesting note is that Star-random performs comparably to original Star when threshold σ is the average similarity of documents in the collection. This further proves our suspicion that degree may not be the best metric.

In figure 17 we see that our proposed algorithms with the exception of Star-markov perform as efficiently as original Star. Star-markov takes longer time as it uses matrix calculation to compute random walk. As expected, Star-lb takes the longest time due to its expensive computation.

From figure 16 and 17 we can conclude that (1) since Star-random is more efficient and achieves comparable F1 to original Star, using degree to pick stars may not be the best metric, (2) since Star-ave is more efficient and achieves comparable F1 to Star-lb; Star-ave can be used as a good approximation to Star-lb to maximize the resulting intra-cluster similarity.

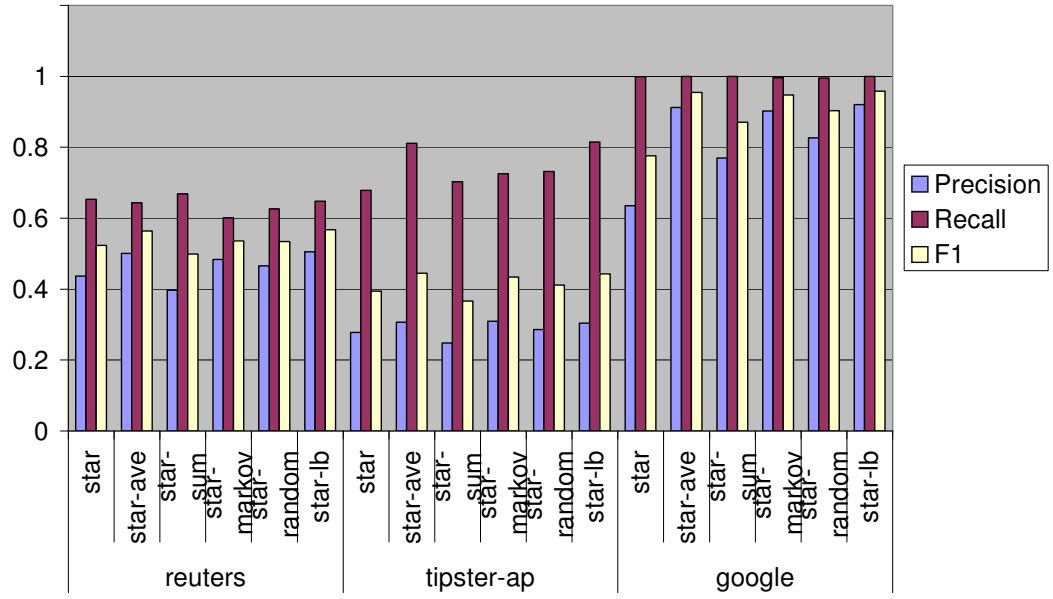

Fig. 16. Effectiveness of Off-line Star Algorithms

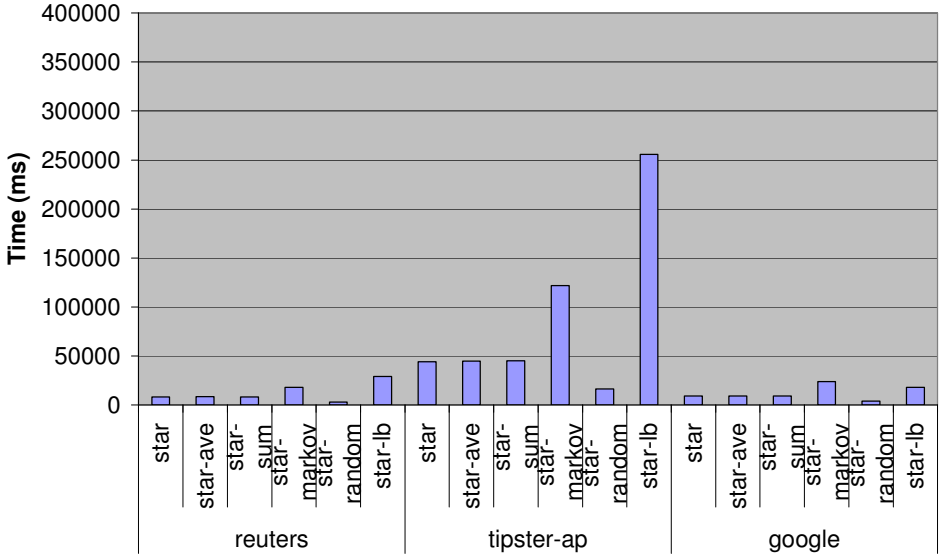

Fig. 17. Efficiency of Off-line Star Algorithms

5.1.2.1.2 Order of Stars

We empirically demonstrate that Star-ave indeed approximates Star-lb better than other algorithms by a similar choice of star centers.

In figure 18 and 19 we present the order in which the algorithms choose their star centers on Tipster-AP and Reuters collections (similar trend is observed on Google collection): from the first star center to the n^{th} star center and where star centers are ranked by their expected intra-cluster similarity (computed from equation 3) from highest to lowest. Star-lb chooses star centers in the order of their expected similarity rank, from highest to lowest.

In figure 18 and 19, we see that Star-ave chooses star centers in an order similar to Star-lb. This is not the case of the other algorithms. Therefore picking star centers in descending order of expected intra-cluster similarity can be approximated simply by picking star center in descending order of average similarity with its adjacent vertices.

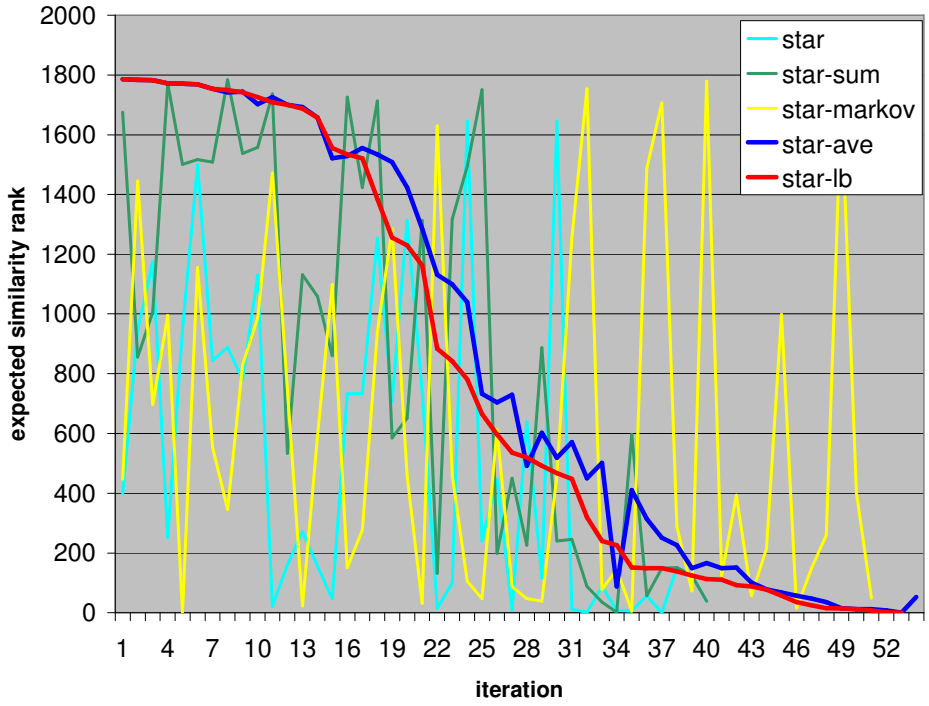

Fig. 18. Order of Stars for Tipster-AP

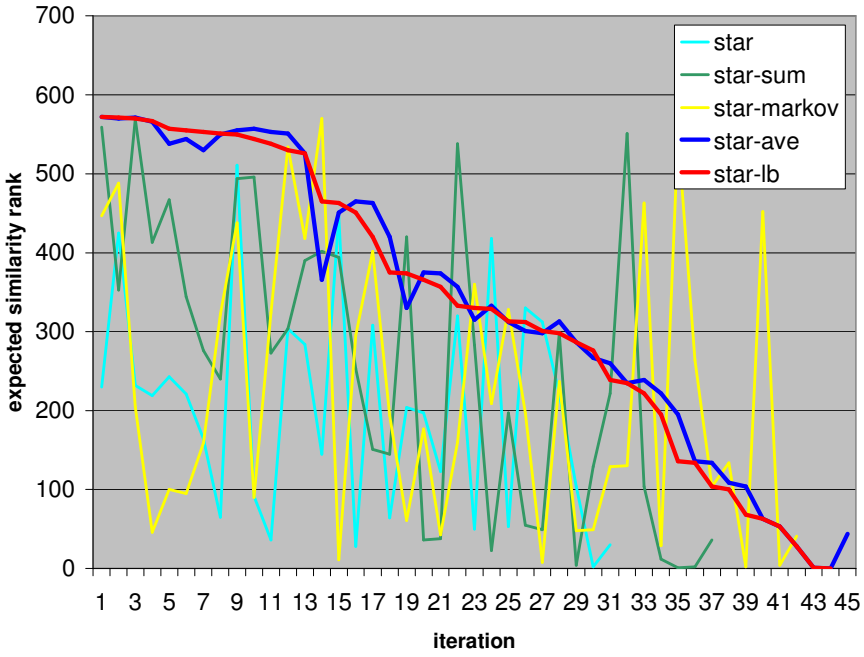

Fig. 19. Order of Stars for Reuters

5.1.2.1.3 Performance of Off-line Algorithms at Different Threshold (σ)

Star clustering has one parameter σ. With a good choice of σ, just enough edges are removed from the graph to disconnect sparsely connected dense sub-graphs. Removing too few edges will group these sparsely connected sub-graphs together; producing high recall but low precision clusters. Removing too many edges will break these dense sub-graphs into smaller, perhaps not-so-meaningful components; producing low recall but high precision clusters.

Figure 20 illustrates the empirical performance evaluation of our proposed off-line algorithms at different threshold values σ on Reuters collection (similar observation is found on Tipster-AP and Google collections). For a similarity graph G (V, E), we represent σ as a fraction (s) of the average edge weight (E_{mean}) in G, i.e. $\sigma = E_{min} + (s * E_{mean})$; where E_{min} is the minimum edge weight in G.

In figure 20, we see that Star-ave and Star-markov converge to a maximum F1 at a lower threshold than the original Star. Star-ave and Star-markov are able to 'spot' sparsely connected

dense sub-graphs and produce reliable vertex cover even when there are few edges removed from the graph. On the contrary, the original Star algorithm needs to remove more edges from the graph before it is able to produce reliable clusters. The maximum F1 value of Star-ave is also higher than the maximum of the original Star. In figure 20 we see that the F1 values of Star-ave coincide closely with the F1 values of Star-lb at all thresholds. This is further evidence that Star-ave can be used to approximate Star-lb at varying thresholds. In figure 20, we see that the F1 gradient (the absolute change in F1 value with each change in threshold) is smaller (some approaching zero) for Star-ave and Star-markov as compared to the original Star. This small gradient means the value of F1 does not change/fluctuate much with each change in threshold. The smaller F1 gradient means that Star-ave and Star-markov are less sensitive to the change in threshold as compared to the original Star.

From figure 20 we can conclude that: (1) Based on maximum F1 value: our proposed algorithm Star-ave outperforms the original Star, (2) our proposed algorithms: Star-ave and Star-markov are able to produce reliable clusters even at a lower threshold where there are only fewer edges removed from the graph; (3) F1 values of Star-ave coincide closely with F1 values of Star-lb at all thresholds; this is further evidence that Star-ave can approximate Star-lb at any given threshold value.

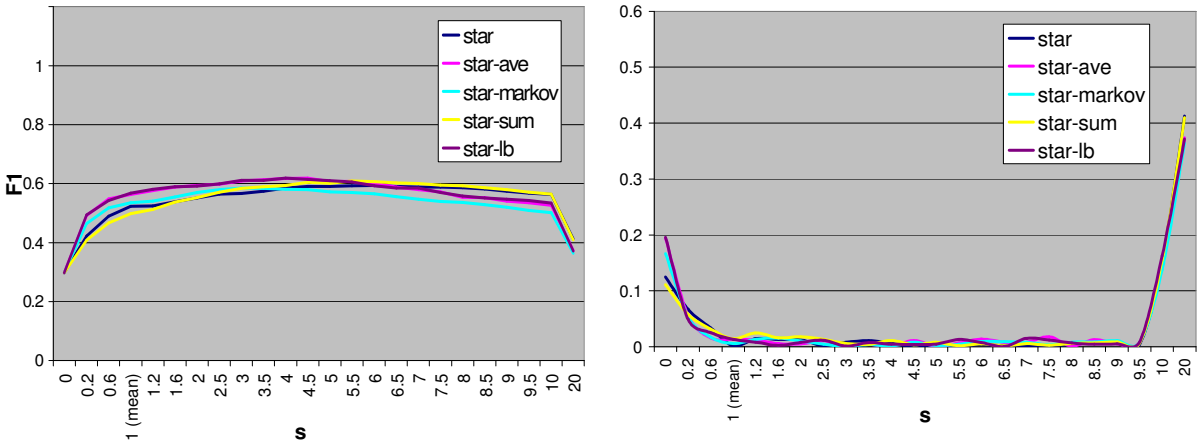

Fig. 20. Performance of off-line algorithms with varying σ on Reuters data

5.1.2.1.4 Performance of Off-line Extended Star Algorithms

We present the results of the experiment that incorporates our idea into the extended Star. In figure 21 and 22, we present effectiveness and efficiency comparison between: our algorithm: Star-ave that has performed the best so far, the original restricted extended star: Star-extended-(r), our algorithms: Star-extended-ave-(r), and Star-extended-sum-(r). The results of comparison with the unrestricted version of extended star give similar findings.

In figure 21, based on F1 values: we see that our proposed algorithm: Star-ave; outperforms Star-extended-(r) on all collections. This is despite the fact that Star-ave uses only very simple idea to incorporate into the original Star. Our proposed algorithm: Star-extended-ave-(r) that incorporates the idea of complement-ave metric to the extended Star algorithms improves the performance of Star-extended-(r).

In figure 22, in terms of efficiency: our proposed algorithms perform comparably or faster than the extended Star; with the exception on Tipster-AP collection (in which Star-ave takes longer time). We believe this difference in time could be because Star-ave picks different Star centers from the extended Star. We also see that incorporating the idea of complement-ave and complement-sum to extended Star does not reduce its original efficiency on all collections.

From figure 21 and 22, we can conclude that: (1) our proposed algorithm: Star-ave obtain higher F1 values than the extended Star on all collections, (2) incorporating the idea of complement-ave metric to the extended Star improves its F1 without affecting its efficiency on all collections.

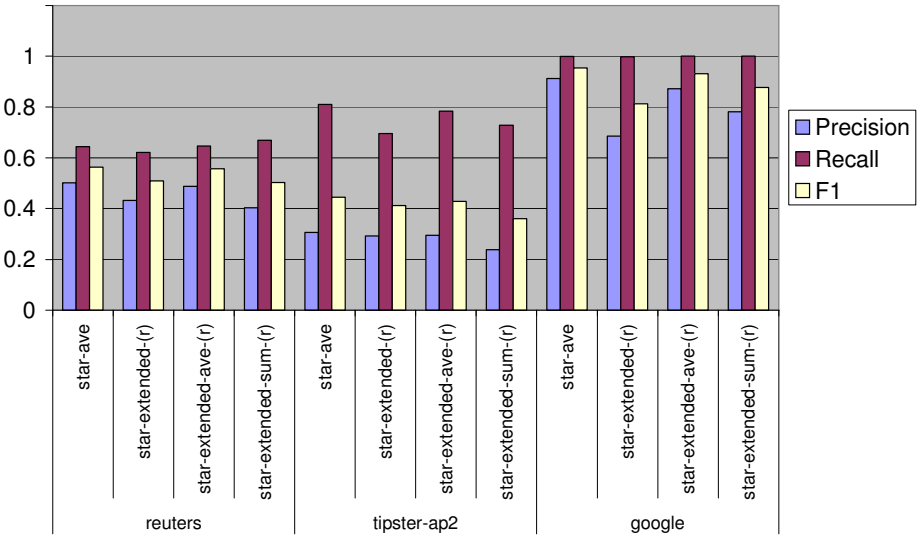

Fig. 21. Effectiveness of Restricted Extended Star Algorithms

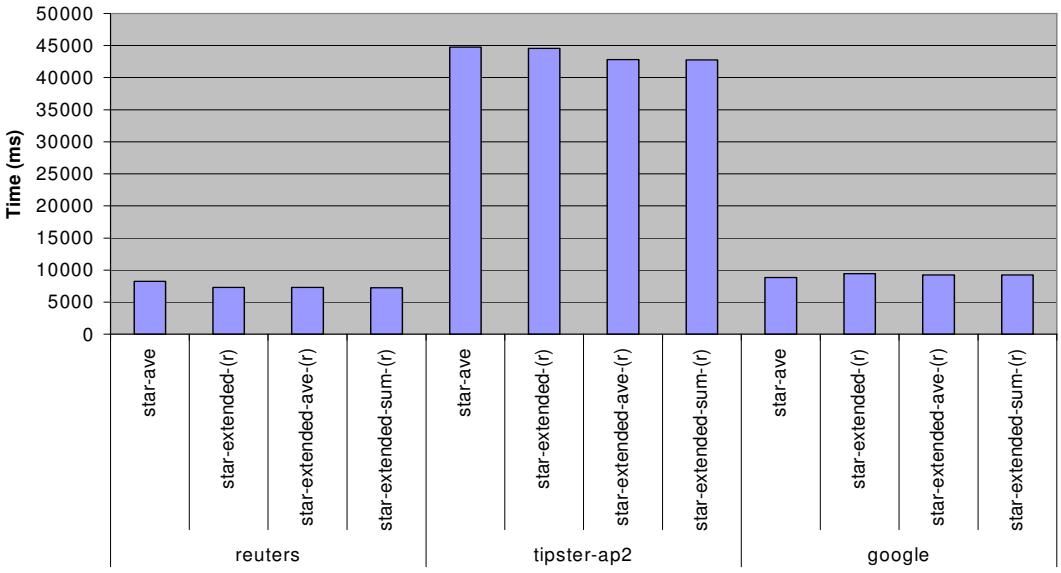

Fig. 22. Efficiency of Restricted Extended Star Algorithms

5.1.2.1.5 Performance of On-line Algorithms

Figure 23 and 24 illustrate the effectiveness and efficiency of the on-line algorithms: the original on-line Star: Star-online, and our on-line algorithms, Star-online-ave and Star-online-sum. We

compare the performance of these algorithms with a randomized version of the algorithm: Star-online-random that picks star centers randomly.

Star-online-ave outperforms Star-online on all collections in terms of F1 (cf. figure 23). Both Star-online-ave and Star-online-sum perform better on all collections than Star-Random. They are more efficient (cf. figure 24) than Star-online on Tipster-AP data and comparable on other collections. From figure 23 and 24, we can conclude that Star-online-ave achieves higher F1 than Star-online. However, due to the fact that Star-online-ave may pick different star centers than Star-online; its efficiency may be affected.

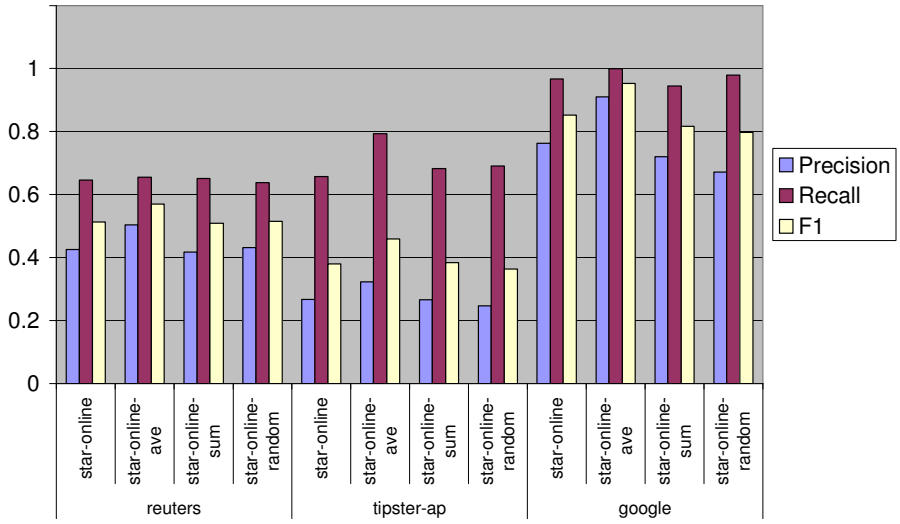

Fig. 23. Effectiveness of On-line Algorithms

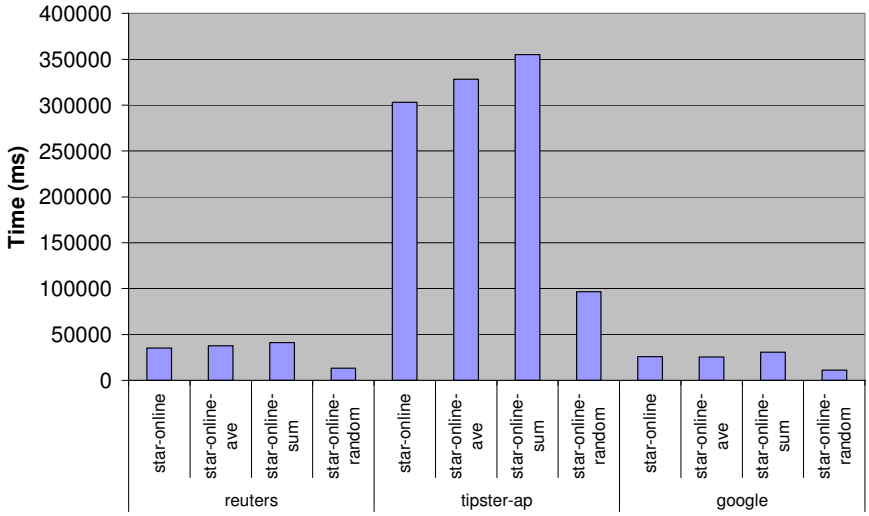

Fig. 24. Efficiency of On-line Algorithms

5.1.2.2 MST-Sim Clustering

Here we present the performance evaluation of our proposed graph-based clustering algorithm: MST-Sim clustering for the task of off-line and on-line document clustering.

When our proposed MST-based clustering algorithm: *MST-Sim* encounters an edge connecting two vertices from different sub-trees, using a similarity metric *Z*, it first calculates the inter-cluster similarity (based on the weights of edges between the clusters) and intra-cluster similarity (based on the weights of edges within the clusters). *MST-Sim* then merges the two clusters only if their inter-cluster similarity is bigger than either one of their intra-cluster similarities times an inflation parameter *c*.

5.1.2.2.1 Inflation Parameters

We evaluate the influence of the inflation parameter *c*. Figure 25 to 27 show the effectiveness (precision, recall, F1 values) of *MST-Sim Kruskal* for varying values of the inflation parameter *c* and similarity metrics on Reuters collection. We observed, but do not report here, a similar trend on the Google and Tipster-AP collections.

As expected, c is a fine tuning parameter: lower values favor recall while higher values favor precision as fewer clusters are merged. Using Beta index (cf. figure 25), $c = 9$ achieves maximum F1 of 59.4%. Using Diameter index (cf. figure 26), $c = 3.5$ achieves maximum F1 of 59.3%. Using Eta index (cf. figure 27), c = 0.8 achieves maximum F1 of 52.3%. From Figure 25 to 27, we can see that using Beta Index and Diameter Index yields the best F1 performance. The same series of experiments, not reported here, for *MST-Sim Borůvka* does not display such clear trends (cf. figure 30). We use (and indicate) in subsequent experiments the value of c yielding the best F1 value.

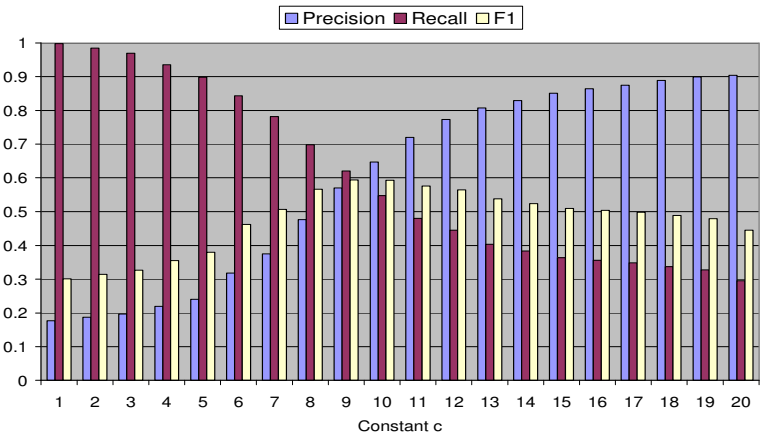

Fig. 25. Effectiveness of MST-Sim (Beta) Kruskal on Reuters

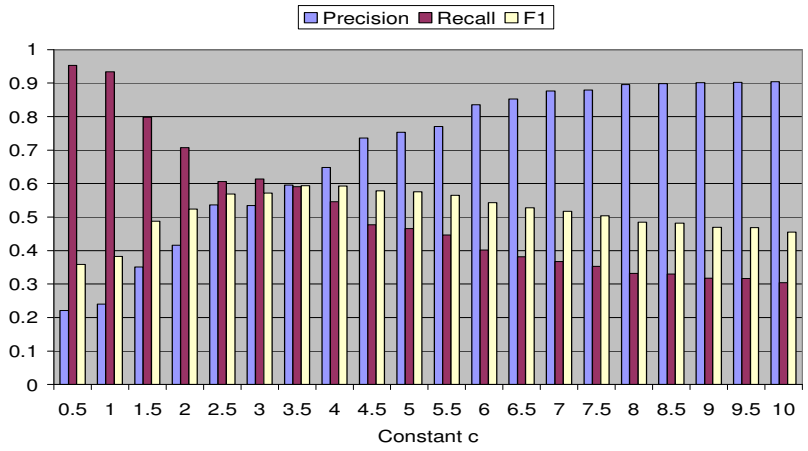

Fig. 26. Effectiveness of MST-Sim (Diameter) Kruskal on Reuters

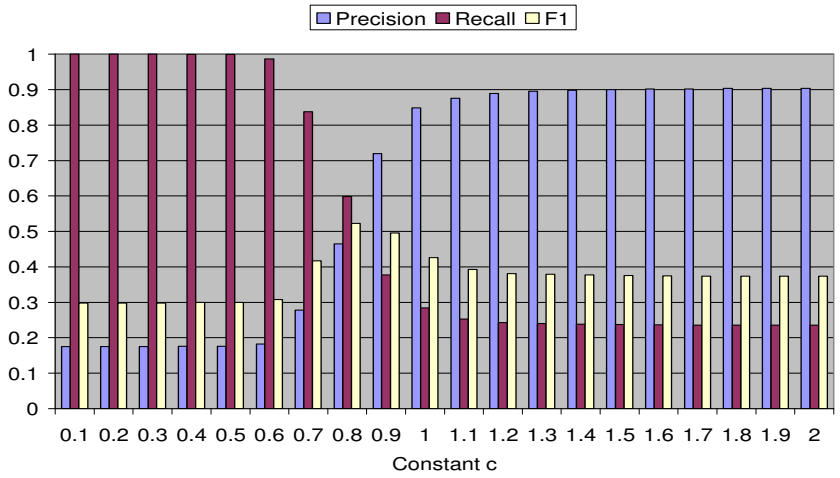

Fig. 27. Effectiveness of MST-Sim (Eta) Kruskal on Reuters

5.1.2.2.2 Kruskal vs. Borůvka

We compare *MST-Sim Kruskal* and *MST-Sim Borůvka*. Figure 28 shows that, in optimal settings, the effectiveness of both algorithms is comparable. Figure 29 shows that *MST-Sim Borůvka* can be faster. However, the performance of *MST-Sim Borůvka* is sensitive to the order it selects vertices and does not follow clear trends for the inflation parameter, as illustrated by figures 30 and 31 (for *MST-Sim Diameter(c) Borůvka*). We therefore subsequently prefer and present results for *MST-Sim Kruskal.*

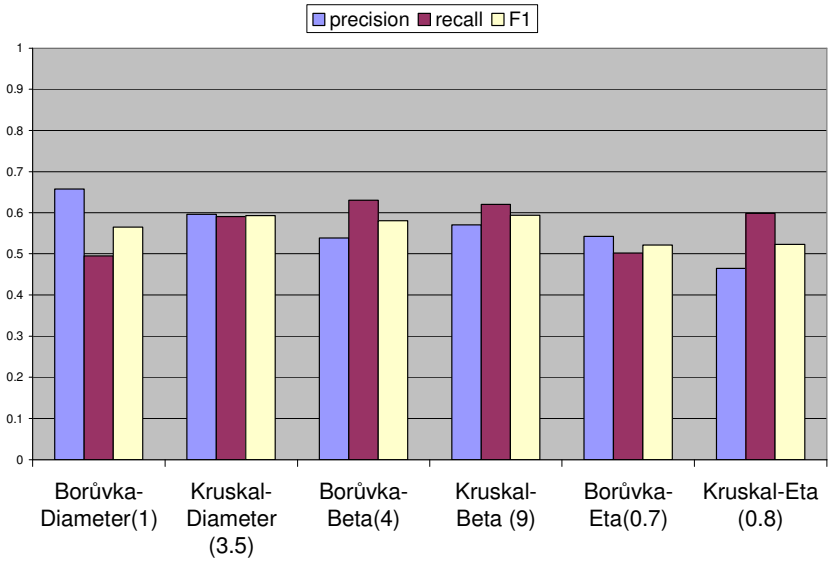

Fig. 28. Effectiveness comparison of Kruskal and Borůvka on Reuters

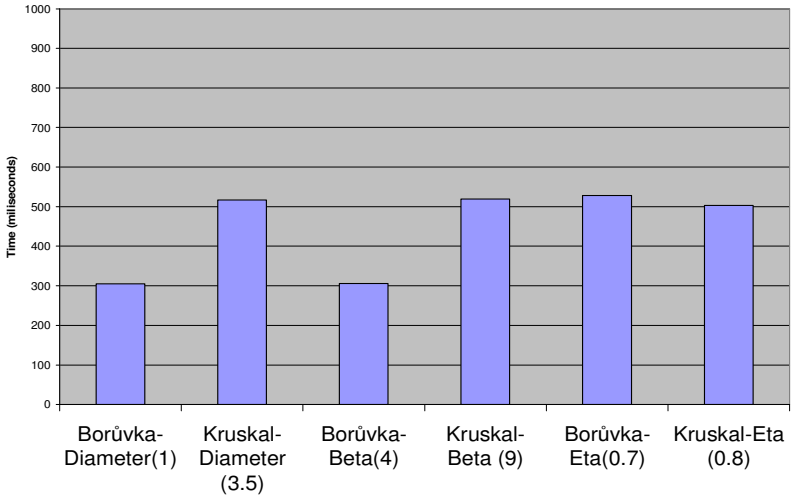

Fig. 29. Efficiency comparison of Kruskal and Borůvka on Reuters

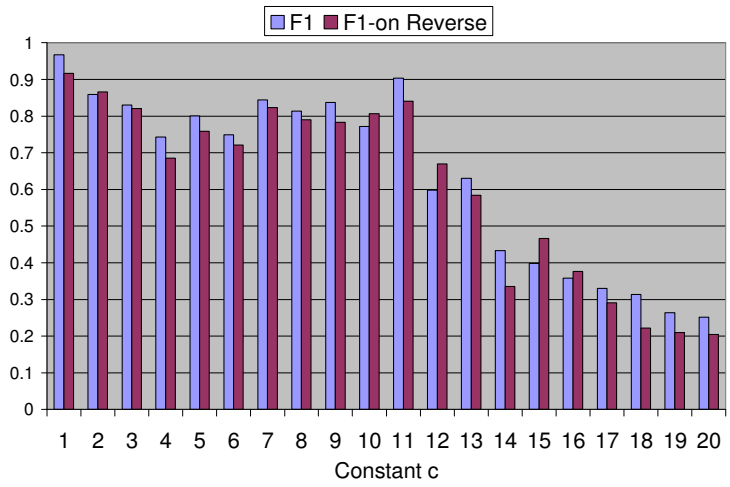

Fig. 30. Effectiveness of Borůvka, different orders of processing vertices on Google

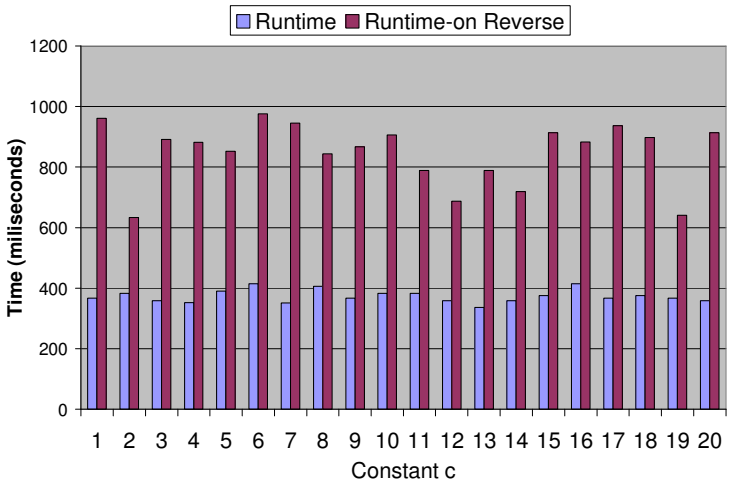

Fig. 31. Efficiency of Borůvka, different orders of processing vertices on Google

5.1.2.2.3 Zahn and Single-link

Our work is motivated by the intuition that we can create agglomerative algorithms that are competitive with existing MST clustering algorithms.

Figure 32 show that, while slightly slower, *MST-Sim Diameter(3.5) Kruskal* is more effective than Zahn's (we vary the value of significance constant for Zahn's and present its best F1

performance) and single-link clustering (with optimum number of clusters given) on Reuters collection. Similar results are observed on Google and Tipster-AP.

On average over all collections, our best performing algorithm: *MST-Sim Kruskal* with Diameter index achieves 11.0% increase in F1 value over the best F1 values achieved by Zahn's and 61.8% increase in F1 value over the F1 values achieved by single-link.

In efficiency (cf. figure 32), our proposed *MST-Sim Kruskal* with Diameter index is slower than Zahn's (we observe that Zahn's efficiency does not change much at different significance constants) and single-link. This is possibly because we use Prim's algorithm (which is more efficient than Kruskal's algorithm) for finding the MST in our Zahn's and single-link implementation.

On average over all collections, our best performing algorithm: *MST-Sim Kruskal* with Diameter index is 18.4% slower than Zahn's and 15.8% slower than single-link. However this loss in efficiency is met with the high gain in effectiveness. Furthermore, as our algorithm requires only local information of the MST, it is more easily adaptable to dynamic or on-line application than Zahn's or single-link that requires the MST to be built before determining edges to prune.

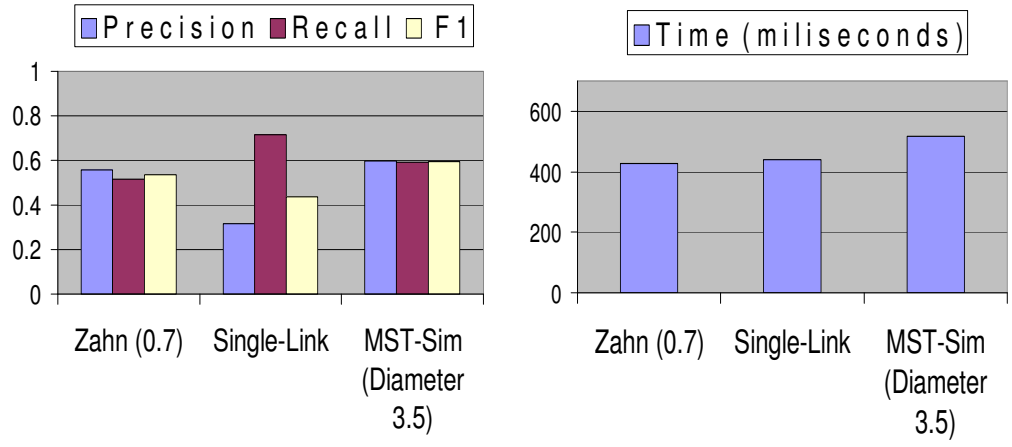

Fig. 32. Effectiveness and Efficiency of Zahn, Single-link and MST-Sim Kruskal on Reuters

5.1.2.2.4 Other State-of-the-Art Algorithms

The following results show that, for the problems at hand, our MST clustering is significantly faster than, yet as effective as, other state of the art clustering algorithms such as K-means, Markov clustering, and Star clustering. In fact, in optimal settings of all the algorithms *MST-Sim Diameter Kruskal* is consistently more effective.

In figure 33 to 39, we compare the effectiveness and efficiency of our proposed algorithms: *MST-Sim Kruskal* using Eta, Beta and Diameter index with K-means (we use an efficient variant of K-means, i.e. K-medoids, with the optimum number of clusters given), Markov Clustering (we vary MCL's inflation parameter and present its best F1 performance) and Star Clustering (with the value of threshold as the average similarity of documents in the given collection) on Reuters, Tipster-AP and Google.

From figures 33 to 35, we see that *MST-Sim* is more effective than Star Clustering in precision and F1 on all collections. When compared to the best performance of Markov Clustering (MCL), *MST-Sim* using Diameter Index is equally or more effective in recall and F1 on all collections. When compared to K-medoids, *MST-Sim* using Diameter index is more effective in recall and F1 on all collections. On average over all collections, our best performing algorithm: *MST-Sim* using Diameter index achieves 37.1% improvement in F1 over Star clustering, 0.93% improvement in F1 over MCL, 21.1% improvement in F1 over K-medoids.

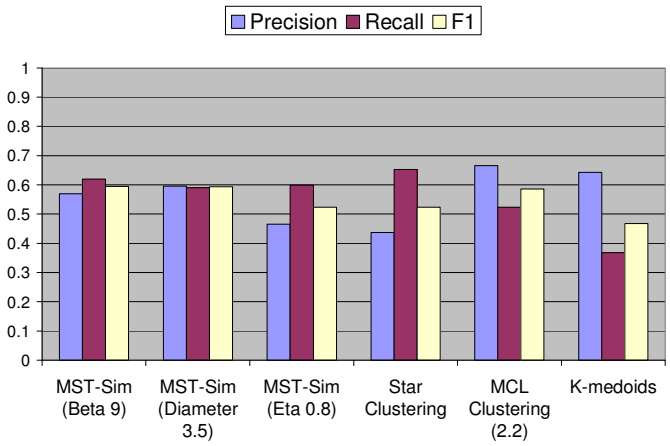

Fig. 33. Effectiveness comparison of off-line algorithms on Reuters

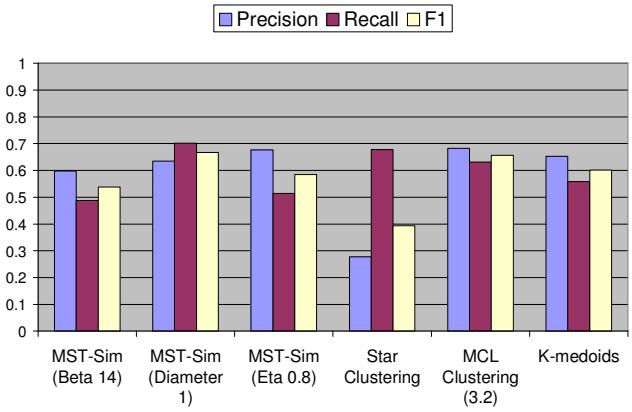

Fig.34. Effectiveness comparison of off-line algorithms on Tipster-AP

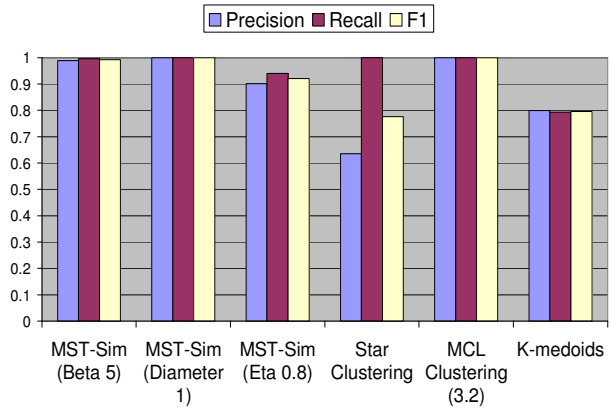

Fig. 35. Effectiveness comparison of off-line algorithms on Google

From figure 36 to 38, we see that all variants of our MST clustering algorithms are faster on all collections than Star Clustering which is slower because it requires calculation of threshold and removal of edges from the graph. Our MST clustering is also faster on all collections than MCL that requires expensive matrix computation. However, our MST clustering is slower than K-medoids that does not need to construct the graph or pre-compute all pair wise similarities.

The fact that graph-based clustering algorithms are slower than K-medoids (due to their computation of all pair-wise similarities) indicates that there needs to be a way to make this pre-processing step more efficient.

Say we have a friendly oracle that can tell us instantly any pair wise similarities between documents; when pre-processing time is not a factor (cf. Figure 39), our proposed algorithms and Star are faster than K-medoids. This shows that it is mostly pre-processing that has adverse effect on the efficiency of graph-based clustering algorithms.

On average over all collections, our best performing algorithm: *MST-Sim* using Diameter index is 15.3% faster than Star clustering, 28.9% faster than MCL, and 41.7% slower than K-medoids.

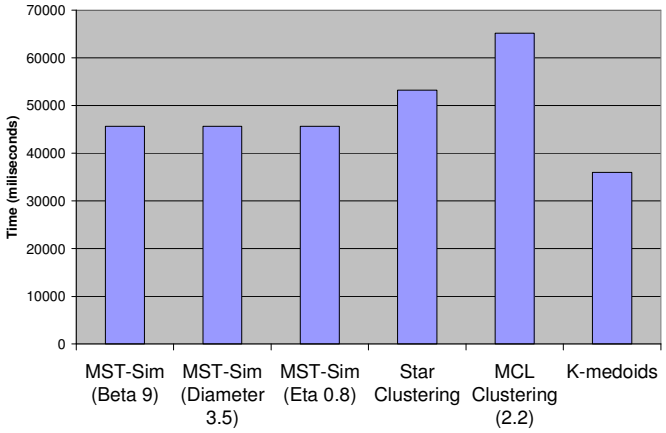

Fig. 36. Efficiency comparison of off-line algorithms on Reuters

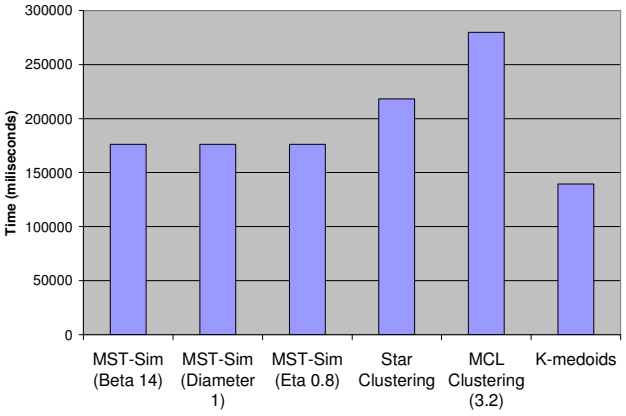

Fig.37. Efficiency comparison of off-line algorithms on Tipster-AP

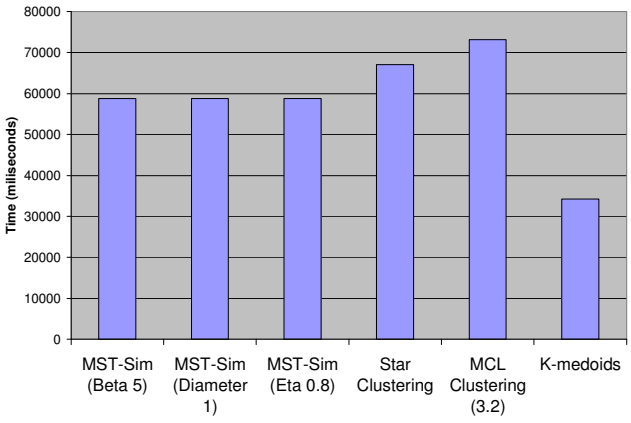

Fig. 38. Efficiency comparison of off-line algorithms on Google

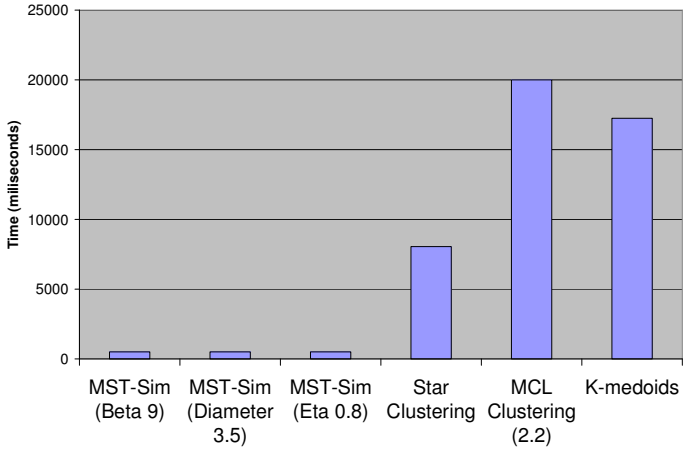

Fig.39. Efficiency comparison of off-line algorithms on Reuters (minus pre-processing)

5.1.2.2.5 On-line Algorithms

In figure 40 and 41 we present the effectiveness and efficiency comparison of our on-line algorithms with the on-line version of Star clustering, i.e. Star-online, on Reuters collection. Similar trend is observed in Google and Tipster-AP data.

In terms of effectiveness (cf. figure 40), the on-line variants of our algorithm are equally or more effective in precision and F1 than Star-online. On average over all collections, the on-line variant of our best performing algorithm: *MST-Sim* using Diameter index achieves 28.6% improvement in F1 over Star-online.

In terms of efficiency (cf. figure 41), the online variants of our algorithm are all faster than Star-online. On average over all collections, the on-line variant of our best performing algorithm: *MST-Sim* using Diameter index is 31.8% faster than Star-online.

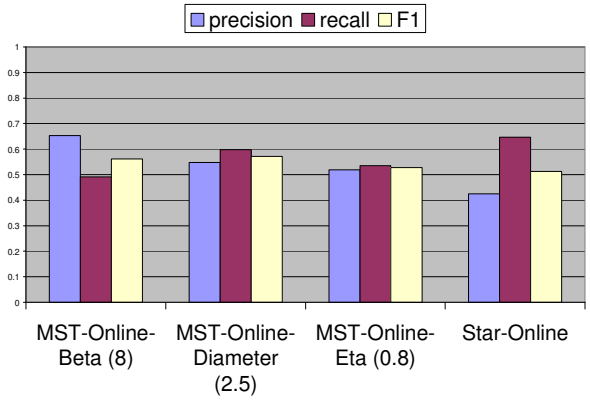

Fig. 40. Effectiveness comparison of on-line algorithms on Reuters

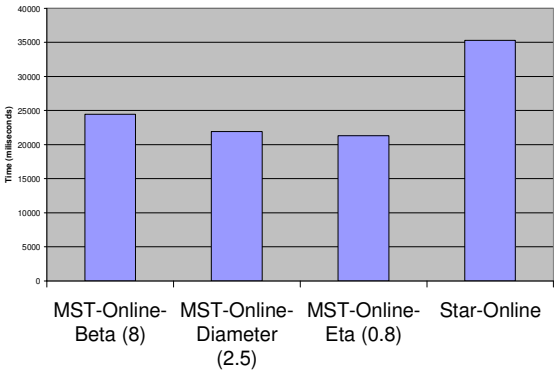

Fig. 41. Efficiency comparison of on-line algorithms on Reuters

5.1.2.3 Ricochet: a Family of Unconstrained Graph-based Clustering

Here we present the performance evaluation of our proposed clustering algorithm: Ricochet for the task of document clustering.

5.1.2.3.1 Apples and Oranges: Comparison with Constrained Algorithms

We first compare the effectiveness and efficiency of our proposed algorithms with those of K-medoids, Star, and (our improved version of Star) Star-ave, in order to determine the consequences of combining ideas from both algorithms to obtained an unconstrained family not requiring parameters. There is, of course, a significant benefit per se in removing the need for parameter setting. Yet the subsequent experiments show that this can be done, only in the some cases, at a minor cost in effectiveness.

In figure 42, we can see that the effect of combining ideas from both K-means and Star in our proposed algorithms improves precision for Google data. Our proposed algorithms also improve the F1-value and maintain recall. In particular, CR is better than K-medoids, Star and Star-ave in terms of F1-value. BSR and OCR are better than K-medoids and Star in terms of F1-value.

In terms of efficiency (cf. figure 43), CR and OCR are faster than Star and Star-ave. K-medoids is much faster than all the graph-based clustering algorithms because it does not require computation of pair wise similarities between all the documents.

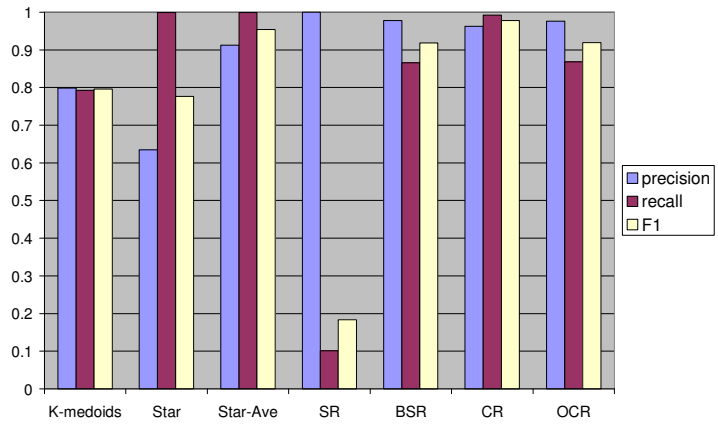

Fig. 42. Effectiveness on Google Data

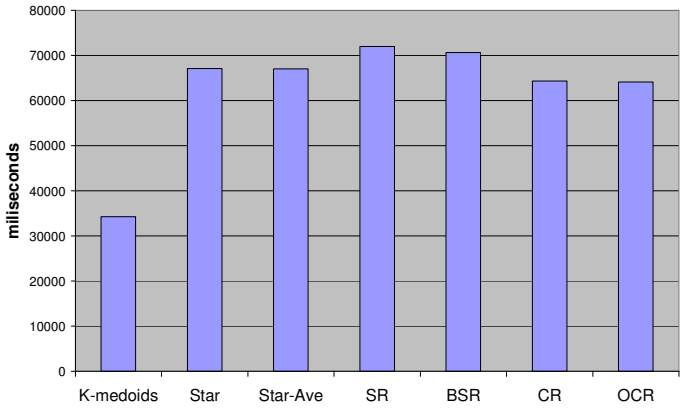

Fig. 43. Efficiency on Google Data

On Tipster-AP data (cf. figure 44), BSR and OCR yield a higher precision and F1-value than Star and Star-Ave. OCR performs the best among our proposed algorithms and its effectiveness is comparable to that of a K-medoids supplied with the correct number of clusters. CR and OCR are also faster than Star and Star-ave (cf. figure 45).

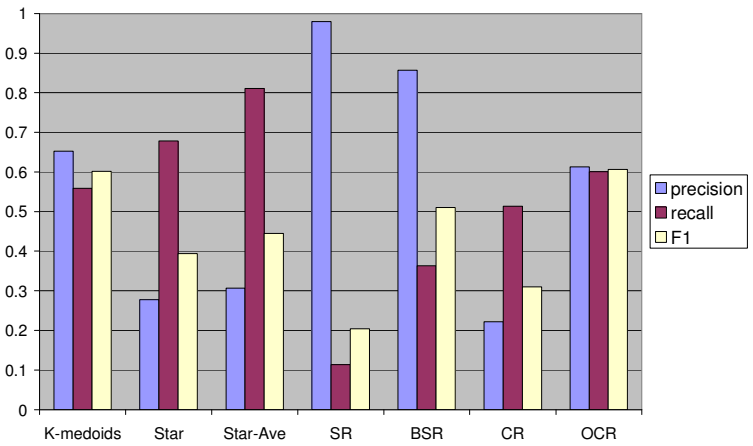

Fig. 44. Effectiveness on Tipster-AP Data

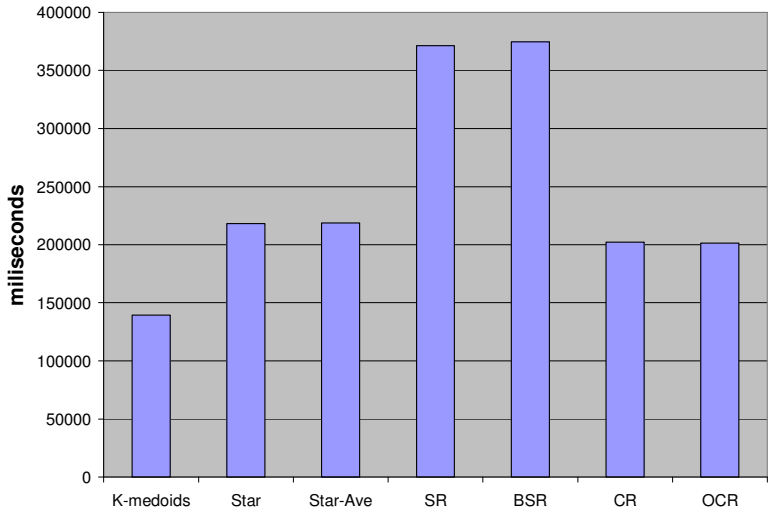

Fig. 45. Efficiency on Tipster-AP Data

On Reuters data (cf. figure 46), In terms of F1-value, it is OCR that performs the best among our proposed algorithms. OCR performance is better than K-medoids and is comparable to that of Star and Star-Ave.

In terms of efficiency (cf. figure 47), OCR is faster than Star and Star-Ave but slower than K-medoids which does not require pair wise similarity computation between all the documents.

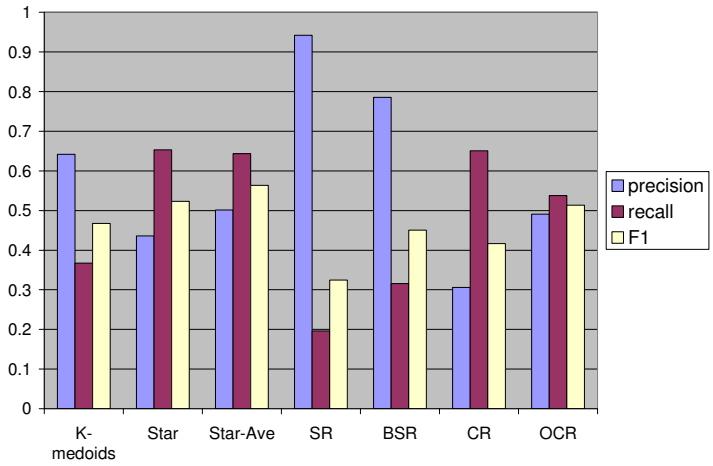

Fig. 46. Effectiveness on Reuters Data

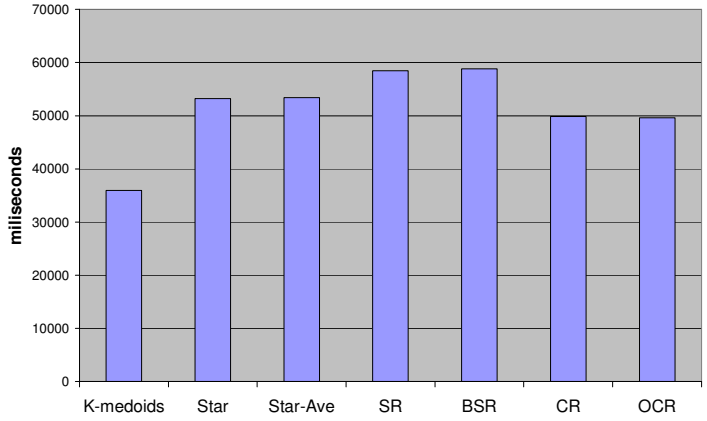

Fig. 47. Efficiency on Reuters Data

In summary, BSR and OCR are the most effective among our proposed algorithms. BSR achieves higher precision than K-medoids, Star and Star-Ave on all three data sets.

We believe BSR achieves higher precision because at each step it tries to maximize the average similarity between vertices and their centroids through reassignment of vertices. Choices of centroids that are of heavy weights and as far away from each other as possible may also have positive effects on the purity of clusters produced.

OCR achieves a balance between high precision and recall, and obtains higher or comparable F1-value than K-medoids, Star and Star-Ave on all the data sets. In terms of F1 value, OCR is 8.7% better than K-medoids, 23.5% better than Star, and 7.9% better than Star-Ave.

In terms of efficiency, the sequential algorithm, SR and BSR are not efficient. The concurrent algorithms CR and OCR are more efficient than Star, Star-Ave and our other proposed algorithms. OCR is 6.3% faster than Star, 6.5% faster than Star-Ave and 56.5% slower than K-medoids (due to pre-processing time).

When pre-processing time is not factored in (cf. figure 48 to 50), the graph-based algorithms: *Star*, *Star-Ave*, *CR*, *OCR* are all faster than K-medoids. This shows that it is mostly pre-processing that has adverse effect on the efficiency of graph-based clustering algorithms.

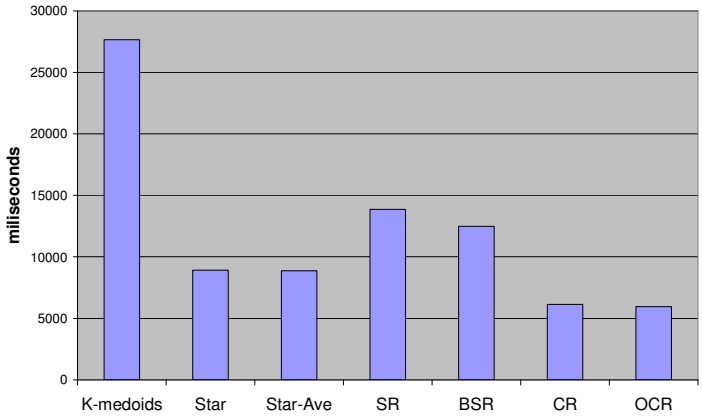

Fig. 48. Efficiency on Google (minus pre-processing)

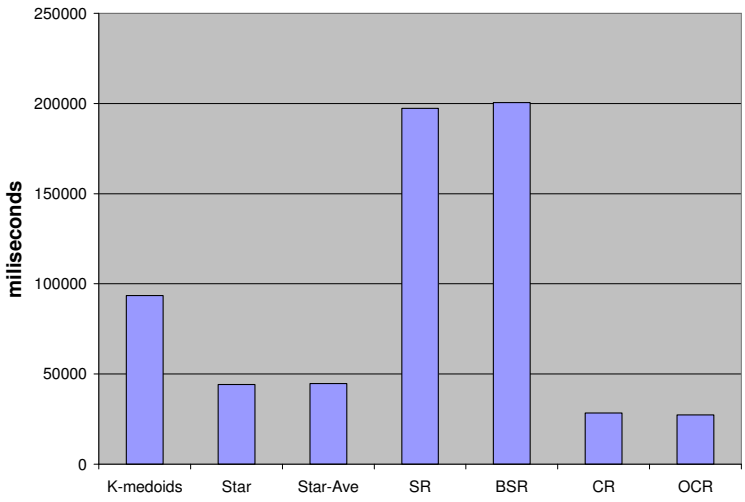

Fig. 49. Efficiency on Tipster (minus pre-processing)

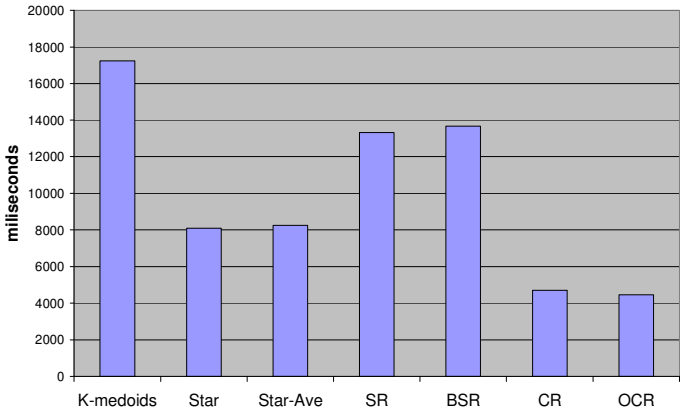

Fig. 50. Efficiency on Reuters (minus pre-processing)

5.1.2.3.2 Comparison with Unconstrained Algorithms

Here we present the comparison of performance of our proposed algorithm: Ricochet, which is unconstrained, with another unconstrained graph-based clustering algorithm: Markov clustering. Markov clustering does not require explicit parameters like number of clusters or threshold to be defined a priori. It however requires an implicit, fine tuning parameter called inflation parameter to be defined a priori.

We first illustrate the influence of Markov clustering (MCL)'s inflation parameter on the algorithm's performance. We vary it between 0.1 and 30.0 (we have empirically verified that this range is representative of MCL performance on our data sets) and report results for representative values.

As shown in figure 51, at a value of 0.1, the resulting clusters have high recall and low precision. As the inflation parameter increases, the recall drops and precision improves, resulting in higher F1-value. At the other end of the spectrum, at a value of 30.0, the resulting clusters are back to having high recall and low precision again.

In terms of efficiency (cf. figure 52), as the inflation parameter increases, the running time decreases, indicating that MCL is more efficient at higher inflation value.

From figure 51 and 52, we have shown empirically that the choice of inflation value affects the effectiveness and efficiency of MCL algorithm. Both MCL's effectiveness and efficiency vary significantly at different inflation values. The optimal value seems to always be around 3.0.

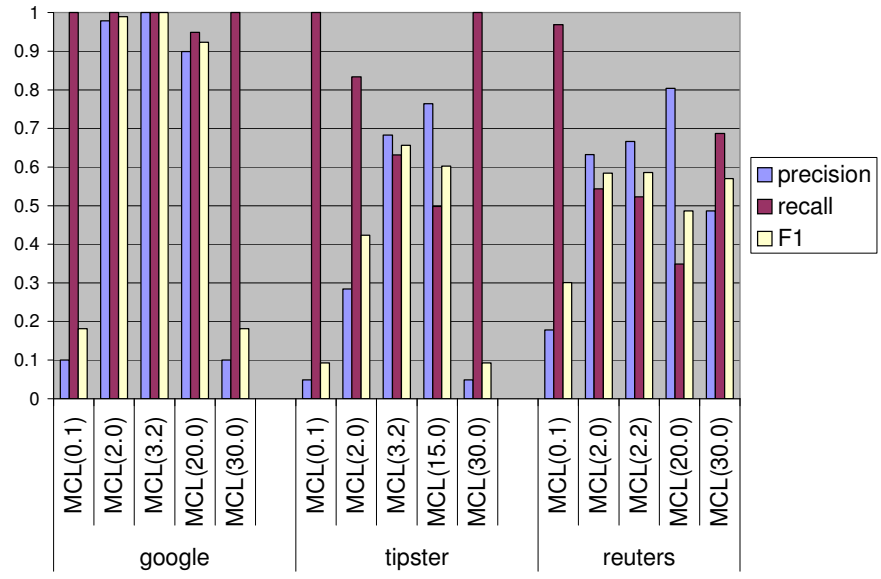

Fig. 51. Effectiveness of MCL at different parameters

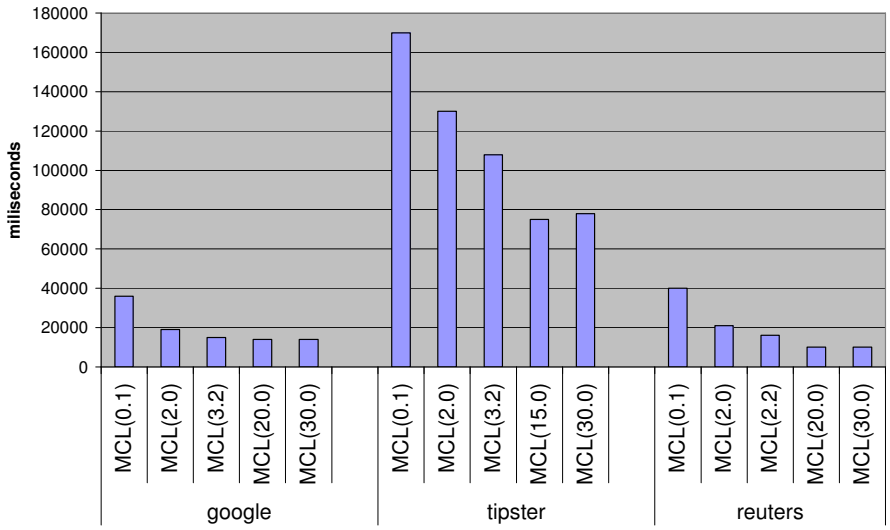

Fig. 52. Efficiency of MCL at different parameters

We now compare the performance of our best performing algorithms, BSR and OCR, to the performance of MCL algorithm at its best inflation value as well as at its minimum and maximum inflation values, for each collection.

From figure 53, on Google data we can see that effectiveness of BSR and OCR is competitive although not equal to that of MCL at its best inflation value. Yet they are much more effective than MCL at the minimum and maximum inflation values. We also see in figure 54 that both BSR and OCR are significantly faster than MCL at all inflation values.

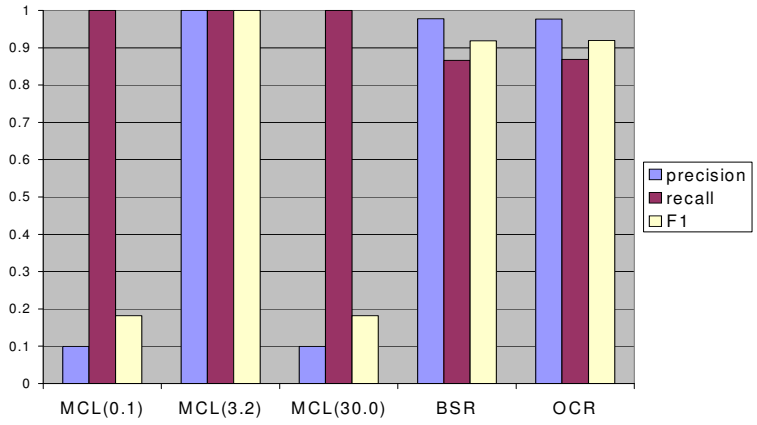

Fig. 53. Effectiveness on Google Data

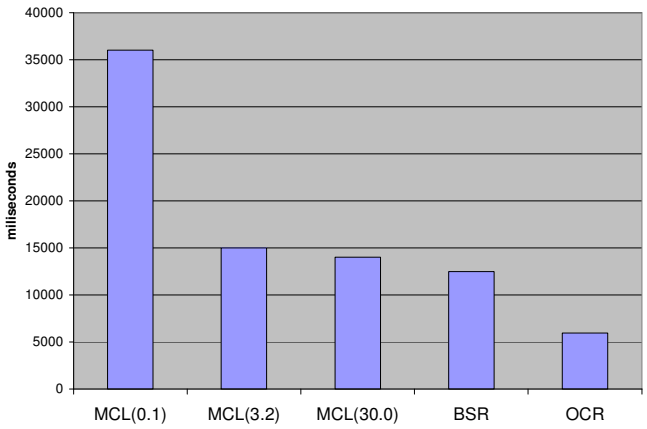

Fig. 54. Efficiency on Google Data

On Tipster-AP data (cf. figure 55), BSR and OCR are slightly less effective than MCL at the best inflation value. However, both BSR and OCR are much more effective than MCL at the minimum and maximum inflation values. In terms of efficiency (cf. figure 56), OCR is also much faster than MCL at all inflation values.

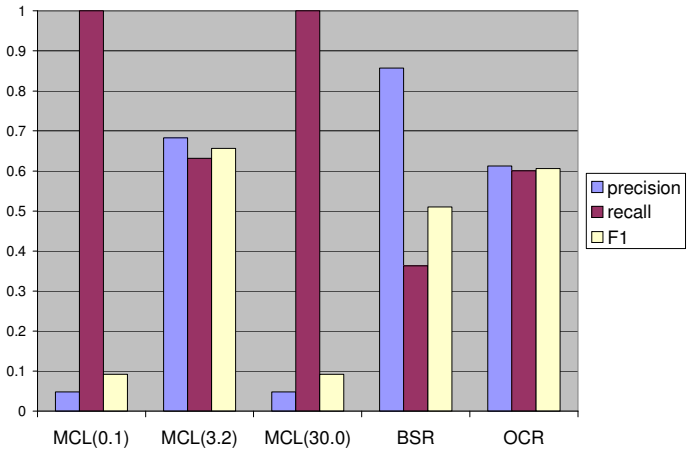

Fig. 55. Effectiveness on Tipster-AP Data

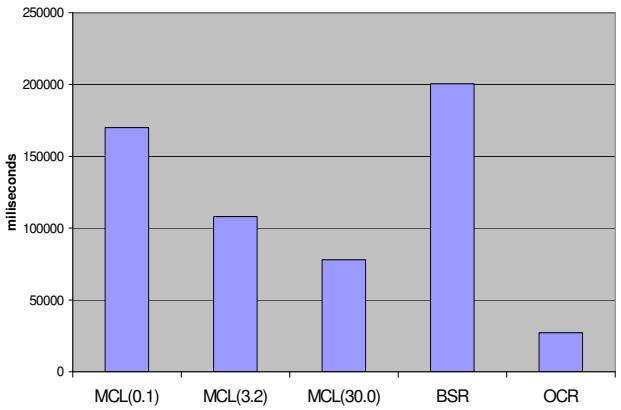

Fig. 56. Efficiency on Tipster-AP Data

The same trend is noticeable on Reuters data (cf. figure 57). BSR and OCR are slightly less effective than MCL at its best inflation value. However, BSR and OCR are more effective than MCL at the minimum and maximum inflation values. In terms of efficiency (cf. figure 58), OCR is much faster than MCL at all inflation values.

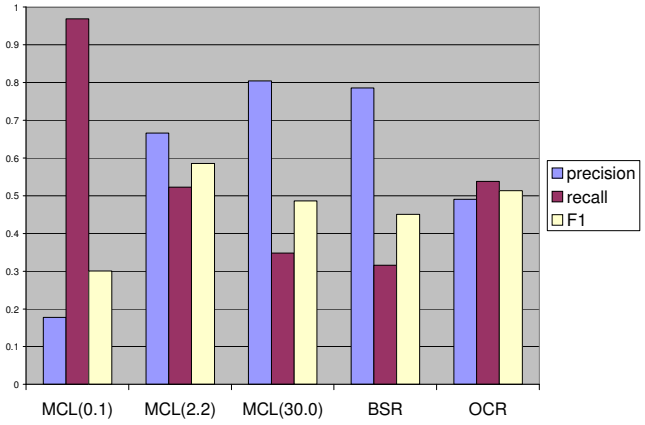

Fig. 57. Effectiveness on Reuters Data

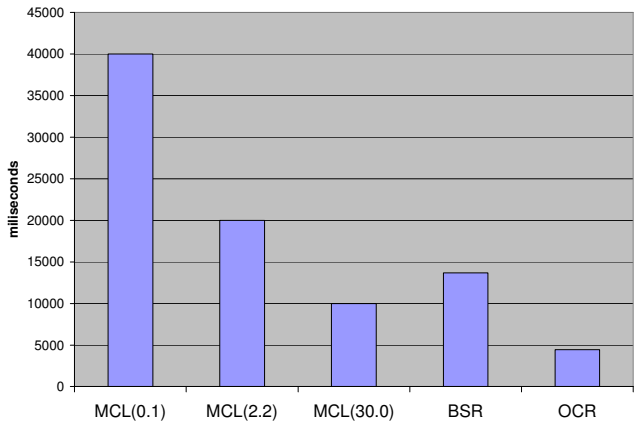

Fig. 58. Efficiency on Reuters Data

In summary, although MCL can be slightly more effective than our proposed algorithms at its best settings of parameter (9% more effective than OCR), one of our algorithms, OCR, is not only respectably effective but also significantly more efficient (70.8% more efficient). Furthermore OCR does not require the setting of any fine tuning parameters like the inflation parameter of MCL that, when set incorrectly, can have adverse effect on its performance (OCR is in average, 334% more effective than MCL at its worst setting).

Section 5.2 Applications

5.2.1 Document Clustering

We have presented experimental results of our proposed graph-based clustering algorithms in the application domain of document clustering in section 5.1.2 above.

5.2.2 k-member Clustering

Here we present experimental results of a variant of *MST-Sim* that can be used to solve k-member clustering problem, called *MST-k*, which uses only one condition for merging, i.e. the condition that the size of the resulting merged cluster must be lesser than or equal to k.

From Figure 59, we see that all our proposed algorithms lose higher information loss than k-member greedy algorithm on all k-values. However, in terms of standard deviation, from Figure

60, we see that all our proposed algorithms (except for MST-Eta) achieve comparable tightness to that of k-member greedy algorithm on all k-values. This perhaps shows that most of the values of numerical attributes in our clusters are close to the mean except for a few "renegade" records that cause the range of values, hence the information loss to increase.

From figure 61, we see that all our algorithms are slower than k-member greedy algorithm on all k-values due to their pre-processing time to construct the graph and compute all pair wise similarities between records.

On average over all k-values, our fastest algorithm for k-member clustering: *MST-k* loses 38.9% more information loss than k-member greedy clustering and is 17.1% slower than k-member greedy clustering but achieves comparable (within 7.7% difference in) tightness of clusters. It is also interesting to note that the efficiency of *MST-k* remains constant at different choices of k-values. This is different from k-member greedy clustering whose efficiency changes with the choice of k-value (cf. figure 61).

As with other graph-based clustering algorithms, the pre-processing time is a bottleneck for *MST-k.* When pre-processing time is not factored in, *MST-k* is much faster than k-member greedy clustering (cf. figure 62). In average, it is 94.4% faster when pre-processing time is not a factor. This further highlights the need for ways to reduce the pre-processing time of graph-based clustering algorithms.

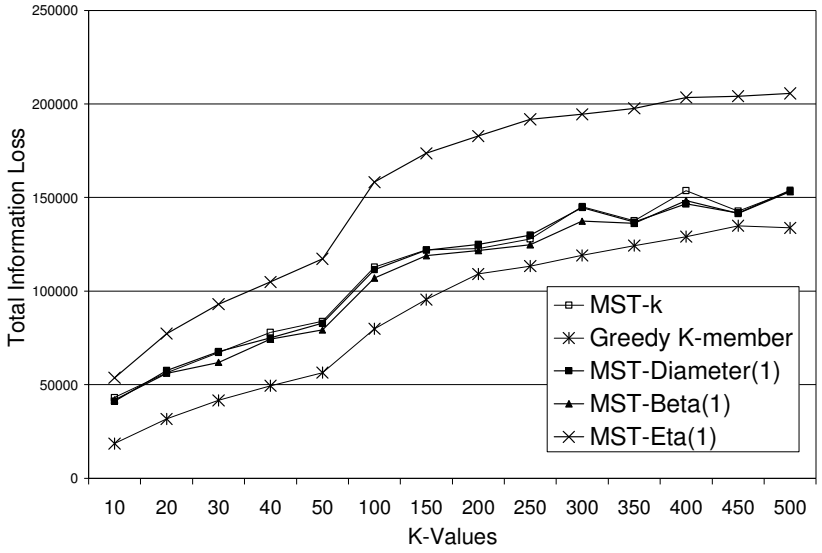

Fig. 59. Information Loss comparison of k-member clustering algorithms

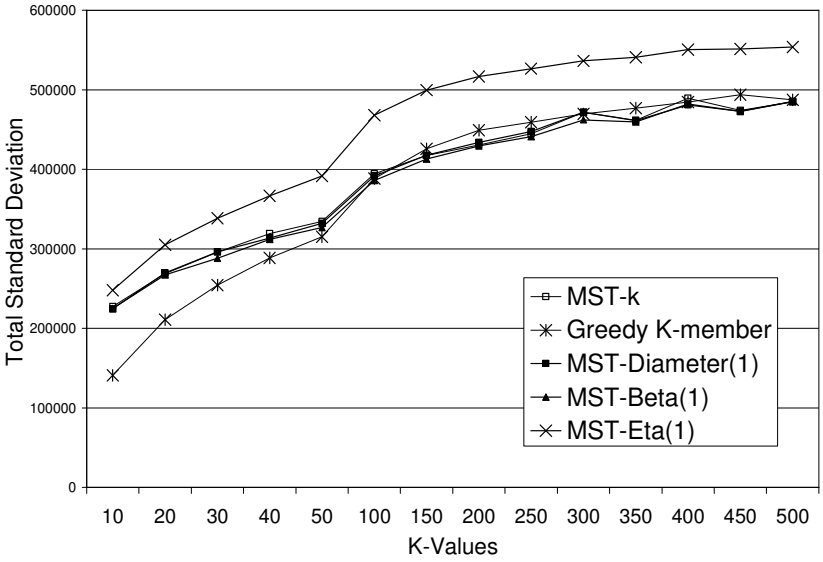

Fig. 60. Standard Deviation comparison of k-member clustering algorithms

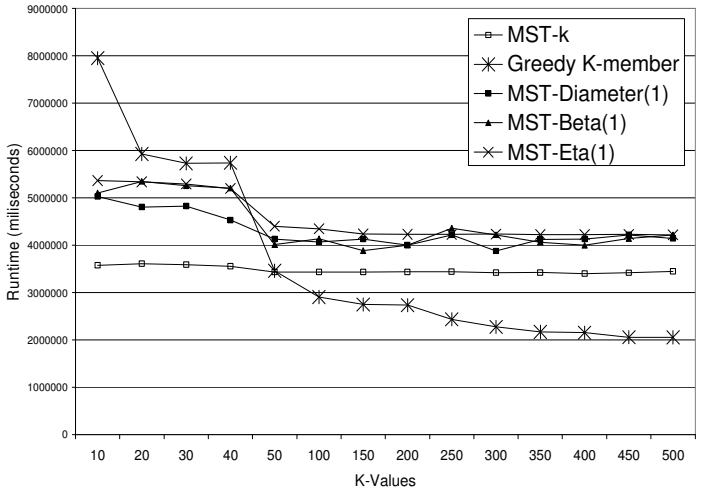

Fig. 61. Efficiency comparison of k-member clustering algorithms

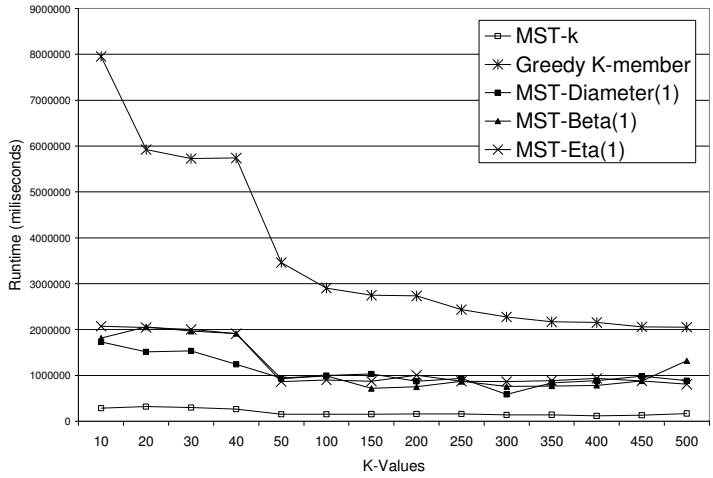

Fig. 62. Efficiency comparison of k-member clustering algorithms (minus pre-processing)

5.2.3 Hierarchical Clustering for POS Tagging of Bahasa Indonesia

5.2.3.1 Experimental Results

Here we present experimental results of our hierarchical clustering for POS tagging of Bahasa Indonesia. We present at each hierarchy level, the weighted-average of precision, recall, and F1-measure produced by each clustering algorithm. We also present results after incorporating users' interactivity.

In figure 63 we compare the weighted-average of precision, recall and F1 produced at different levels by single-link and Borůvka hierarchical clustering.

From figure 63, we can see that in terms of F1 and recall, Borůvka always gives higher F1 and recall than single-link at different levels of clustering. Since Borůvka achieves higher F1 than single-link and allows for ease of users' interactivity, for the remainder of the experimental results we will present Borůvka's results.

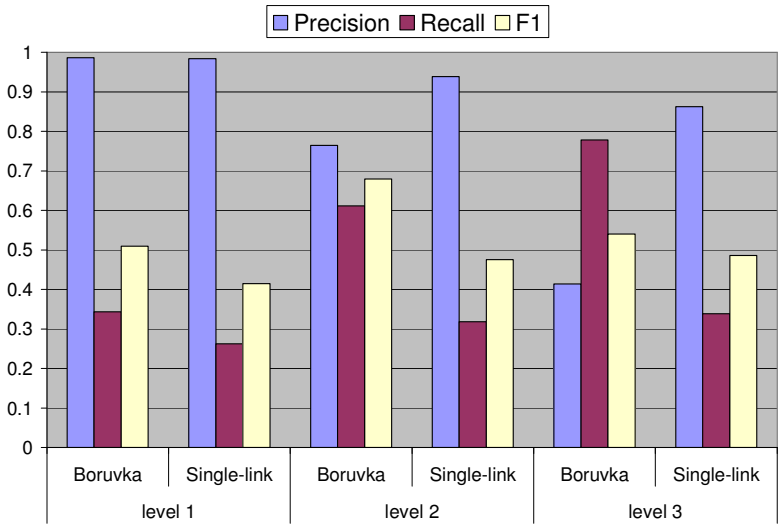

Fig. 63. Borůvka and Single-link Comparison

In figure 64, we compare the weighted-average of precision, recall and F1 produced by Borůvka at different levels of hierarchy.

From figure 64, we can see that precision is highest at level 1 and drops at subsequent levels. Recall is lowest at level 1 and increases at subsequent levels. F1 is highest at level 2. Precision is high at the lowest level when the clusters are small and more pure. Recall is higher at higher level because clusters are merged and bigger clusters are formed. F1, which is a harmonic average of precision and recall, is highest at level 2. This indicates that the best clustering result is formed at level 2.

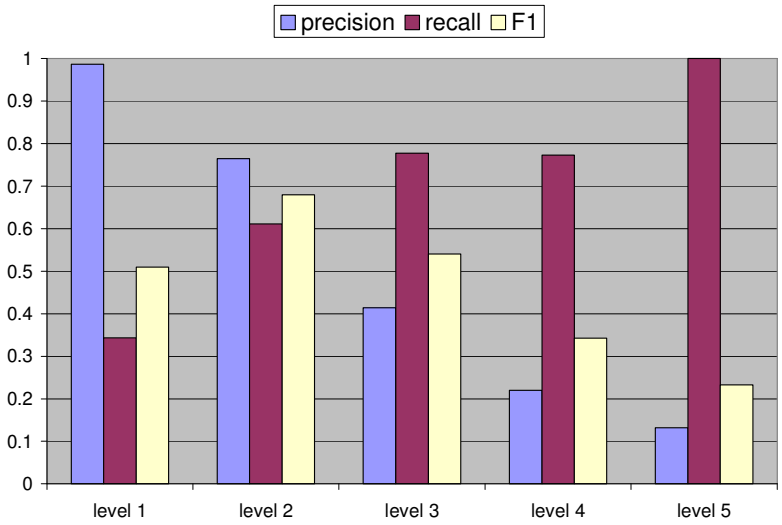

Fig. 64. Borůvka at Different Levels

In figure 65, we present results when users' interactivity in the form of words' constraints (Borůvka-Word), morphological constraints (Borůvka-Morph), and both words and morphological constraints (Borůvka-Word-Morph) is added to level 1 of Borůvka hierarchical clustering.

From figure 65 we can see that adding words and morphological constraints improves both weighted-average of precision, recall, and F1. In particular, adding both words and morphological constraints (Borůvka-Word-Morph) gives the highest improvement in precision, recall and F1 over the original clustering.

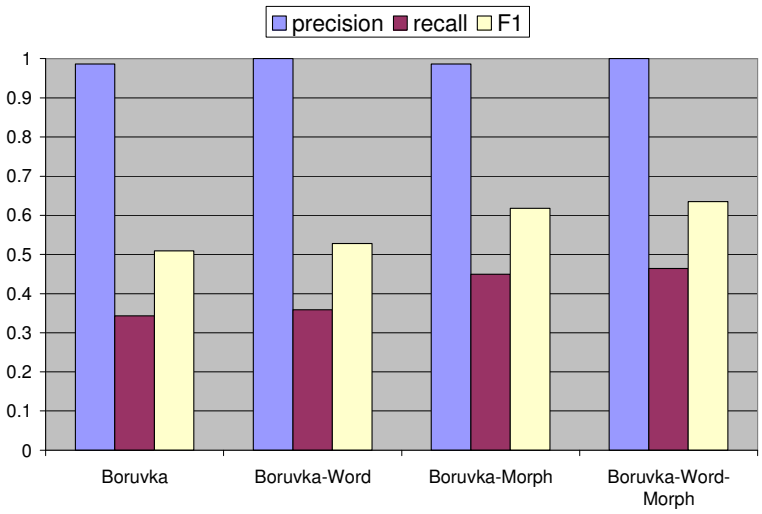

Fig. 65. Adding Constraints to Borůvka (At Level 1)

In figure 66, we compare the weighted-average of F1 values of Borůvka and Borůvka-Word-Morph at different levels of clustering.

From figure 66, we can see that Borůvka-Word-Morph gives higher F1 than Borůvka at different levels of clustering. In particular, Borůvka-Word-Morph gives the highest F1 at level 2 of clustering; indicating that best clustering result is indeed formed at level 2.

Although in this experiment we can achieve optimality at low hierarchy level (i.e. level 2), we believe the small number of words we cluster may contribute to this quickness of merging. When number of words to be clustered is large, the merging may not be so quick and optimality may be achieved only at higher level.

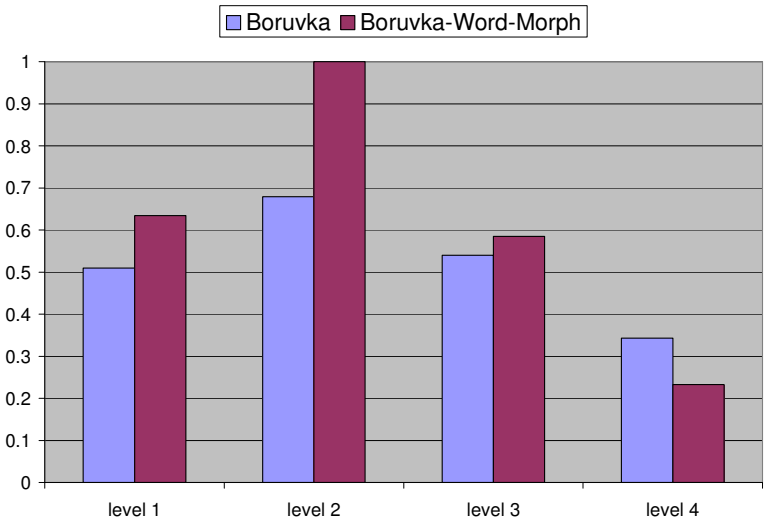

Fig. 66. F1 Values after Adding Constraints to Borůvka (At All Levels)

5.2.3.2 Semantic Senses and Named-Entity Recognition

We observe that clustering at fine granularity displays semantic significance beyond parts-of-speech tagging as previously suggested in [39]. Indeed, it seems that fine granularity clustering does group words by senses and therefore acts as a named-entity recognition algorithm. This is not surprising since morphological and contextual features are used. For example, at level 1 and level 2 names of days, months, years, places, people are grouped in separate clusters (cf. Table 2).

Being morphologically rich, Indonesian language can often present an array of inconsistencies and exceptions [80]. For example, although the affix “ter-” is often used to create adjectives (e.g. terkenal : famous), while some base words (e.g. sangka, sebut) when combined with the affix “ter-” can produce a noun (i.e. tersangka : suspect) or a determinant (i.e. tersebut : the) instead of an adjective. Another example is the word pengadilan (: court) that has the affix “pe-an” commonly used to create nouns. However the word pengadilan is more commonly used in conversation and sentences to refer to the location rather than the substantive concept of a court.

Another example is the repeat words in Indonesian language (e.g. orang-orang : people) that are often used as nouns. However, some base words (e.g. hati, pelan) when repeated produce adjectives (i.e. hati-hati : careful, pelan-pelan : slow) instead of nouns. Faced with these inconsistencies, our clustering method is able to find the correct sense of the word, therefore the correct grouping instead of being sidetracked by the morphological/grammar features (cf. Table 3).

Table 2 Examples of Named-entity Observed

Tag	Examples	Note
NNP	senin selasa rabu kamis jumat sabtu minggu	Days
NNP	januari februari maret april mei juni juli agustus september oktober november desember	Months
NNP	1980 1998 2002 2000 2001	Years
NNP	padang medan surabaya jakarta ambon italia belanda jerman cina swiss jepang as singapura timtim australia malaysia pengadilan kejaksaan	Places
NNP	muzadi wahid sidiq mz bisri	Last name
NNP	hasyim asyawadi cholil abdurahman nur zainuddin	First name

Table 3 Examples of Word Senses Observed

Tag	Examples	Note
DT	ini itu tersebut	this, that, the
JJ	pelan-pelan terbuka lambat hati-hati cepat	
NN	tokoh-tokoh orang-orang	
NNP	padang kejaksaan pengadilan medan surabaya jakarta ambon	places
NN	Tersangka saksi	

5.2.4 Opinion-based Ranking

Here we present results of evaluating our movie ranking against the box office ranking. We use *Top-k* and *Granularity-g* method for evaluating performance. For each of the evaluation, we present metrics for measuring ranking performance. We also present interesting result for the ranking of adjectives by genre.

5.2.4.1 Top-k

We compare the top-k list of our ranking with the top-k list of the box office ranking, for k = 1 to k = 50. For each k, we present *percentage of overlap* and *average rank error*.

Percentage of overlap is measured as the size of overlap (the number of movies in the top-k list of our ranking which are in the top-k list of the box office ranking), divided by k. *Average rank error* is measured as the average of rank differences between the ranks of movies in the top-k list of the box office ranking and their ranks in our ranking.

In figure 67, we present the *percentage of overlap* between the top-k list of our ranking and the top-k list of the box office ranking.

We see that all our methods (except individualNegative, byGenreNegative, and allNegative) perform better (higher *percentage of overlap* with box office ranking) than the ranking from user ratings. individualPositive and byGenrePositive perform the best, achieving more than 70% of overlap with box office ranking for almost all k.

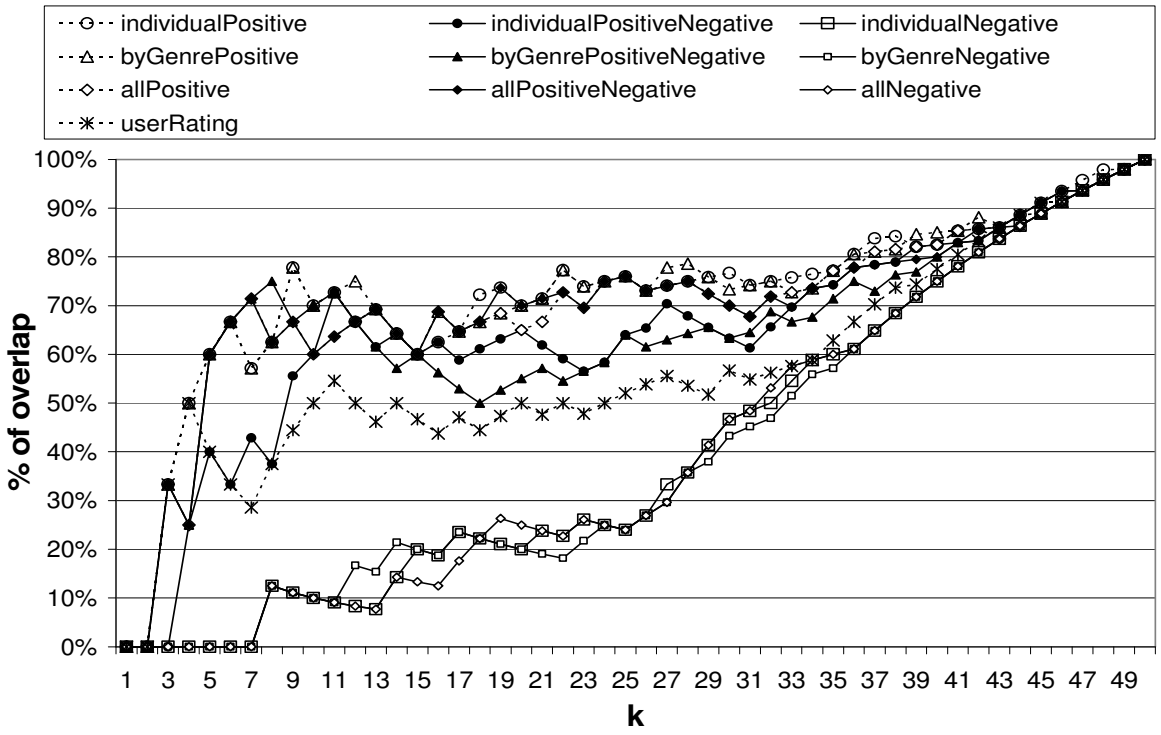

Fig. 67. Percentage of Overlap in top-k Movies

In figure 68 we compare the *average rank error* between the ranks of movies in the top-k of box office ranking and their ranks in our ranking.

From figure 68 we can see that all our methods (except individualNegative, byGenreNegative, and allNegative) perform better (lower *average rank error*) than the ranking from user ratings. individualPositive and byGenrePositive perform the best.

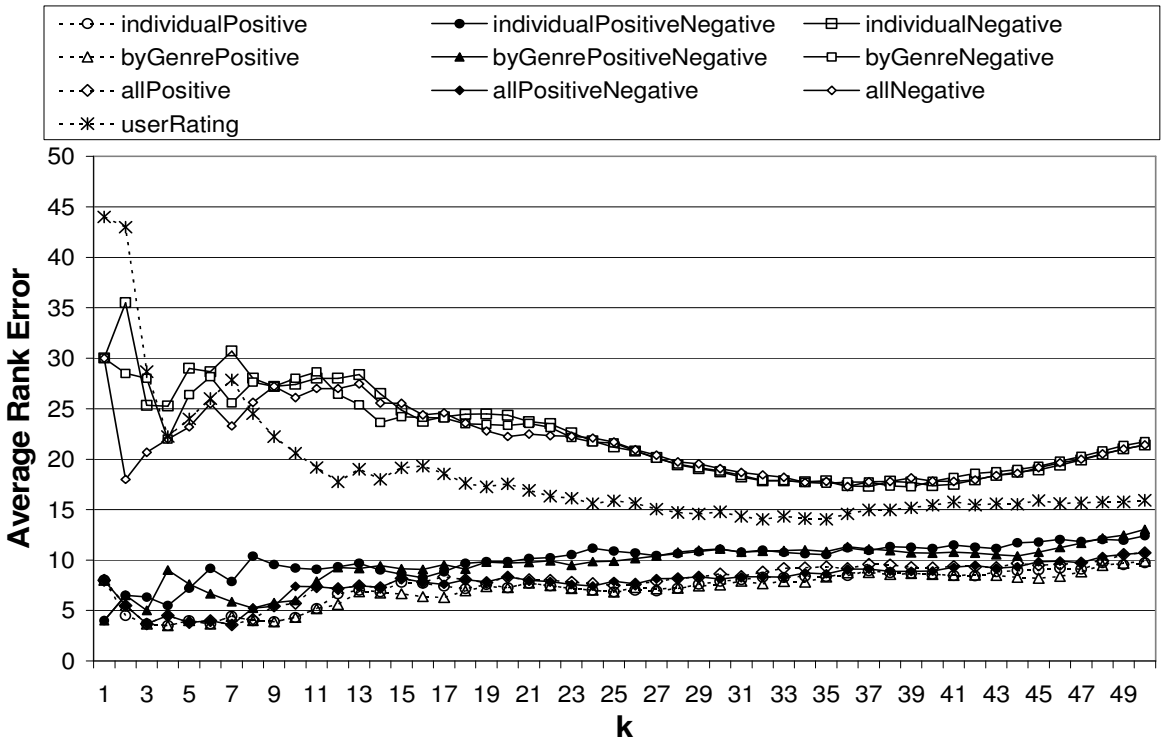

Fig. 68. Average Rank Error in top-k Movies

From these results, we observe that methods which use negative semantic orientation scores are not as effective as those with just the positive semantic orientation scores. This highlights the question on how best to perceive and use the negative semantic orientation scores. Non-negative orientation may not always mean positive orientation, and positive and negative semantic orientation scores may not always combine in a linear fashion.

From these results, we also observe that methods which use reviews grouped by genre (byGenre_) perform better than the methods which combine all the reviews (all_). This maybe because, when we combine all reviews, we lose some information on the genre-dependent context of the adjectives. Such context maybe important in determining the semantic orientations of adjectives which depend on genre: e.g. funny – which maybe positive in comedy genre but negative elsewhere, scary – which maybe positive in horror genre but negative elsewhere, etc.

From these results, we also observe that all our methods (except individualNegative, byGenreNegative, and allNegative) perform better than user ratings in inferring the box office ranking, which we believe to be an objective measure for ranking opinions about the movies. Our results confirm similar observations in [58, 59].

5.2.4.2 Granularity-g

We group movies at different granularity g, for g = 1 (one movie in a group) to g = 50 (all movies in one group). After grouping, movies in the same group are assigned the same rank. Hence, different grouping results in different ranking. For each g, we present *percentage of rank overlap* and *average rank error* with the information loss incurred from coarser ranking.

Percentage of rank overlap is measured as the percentage of movies out of all movies in our dataset which has the same numerical rank in our ranking as in the box office ranking. *Average rank error* is measured as the average of rank differences between the rank we produce for each movie and the movie's box office rank. Information loss is measured as the sum of information loss of each movie. The information loss of each movie is the range of box office figures in its group divided by the maximum range of box office figures of all movies in our dataset.

In figure 69 we present the *percentage of rank overlap* vs. information loss at different granularity g when we compare our ranking to box office ranking.

In figure 69 we can see that all our methods (except individualNegative, byGenreNegative, and allNegative) perform better (higher *percentage of rank overlap* for the same granularity g) than the ranking from user ratings. individualPositive and byGenrePositive perform the best.

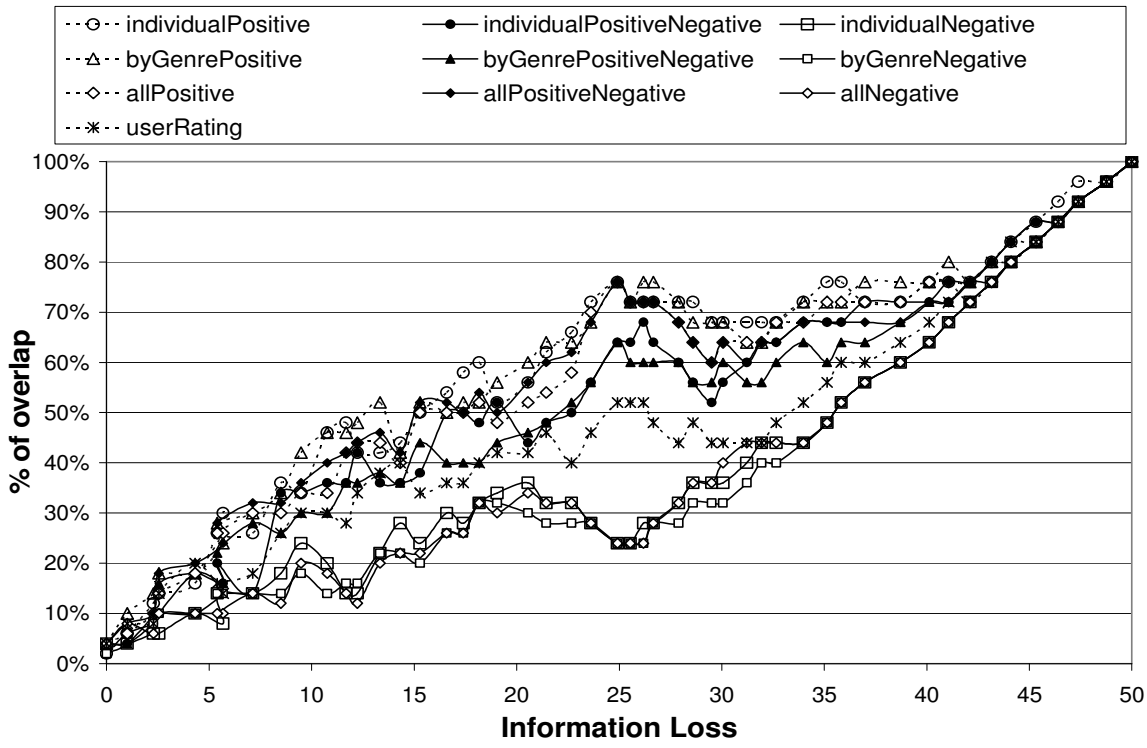

Fig. 69. Percentage of Rank Overlap vs. Information Loss at Different Granularity g

In figure 70 we present the *average rank error* vs. information loss at different granularity g when we compare our ranking with the box office ranking.

When g = 1 (each movie is a group of its own), the information loss is zero and the *average rank error* is maximum. As we group movies (g > 1), the *average rank error* decreases more rapidly than the increase in information loss. This shows that grouping movies can improve the ranking result greatly without incurring too much information loss.

When we zoom in on figure 70, we see that all our methods (except individualNegative, byGenreNegative, and allNegative) perform better (lower *average rank error* for the same granularity g) than the ranking from user ratings. individualPositive and byGenrePositive perform the best.

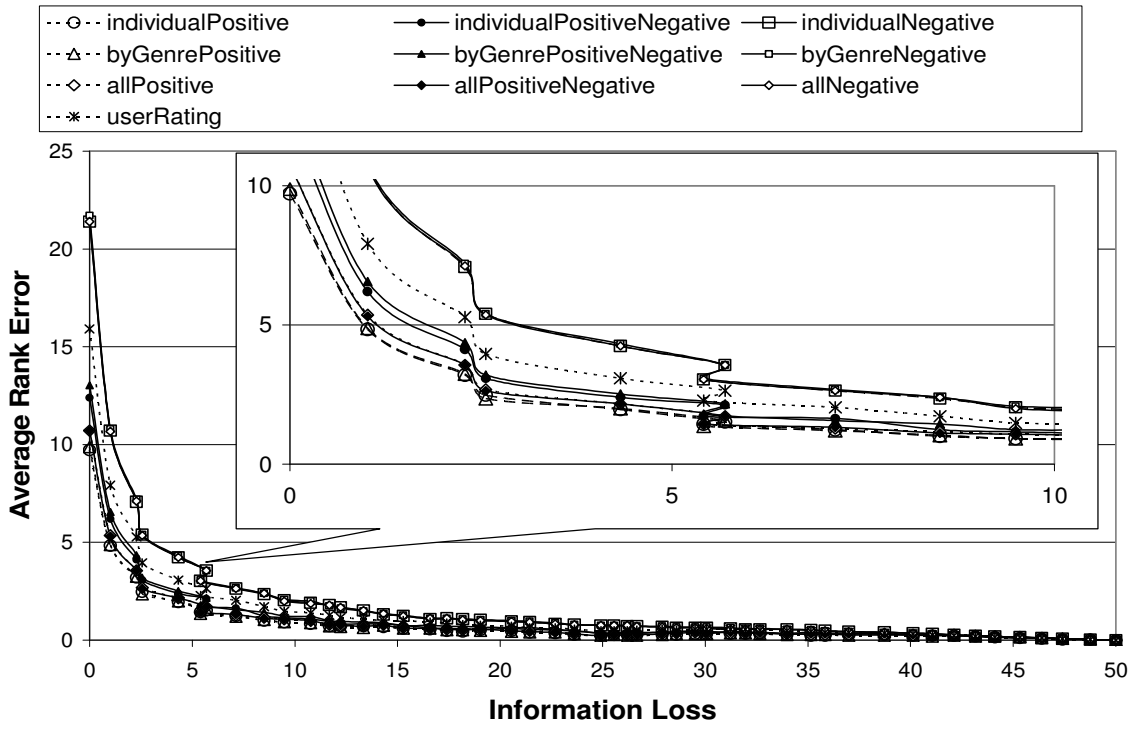

Fig. 70. Average Rank Error vs. Information Loss at Different Granularity g

5.2.4.3 Sensitivity to Starting Adjectives

Here we present the results of evaluating the sensitivity of our method to the starting adjectives we define in Good_Adjectives and Bad_Adjectives sets. We present results of ranking by one of our best performing method (individualPositive) as compared to the box office ranking.

In figure 71, we present the results of running individualPositive method with different number of starting adjectives extracted from our Good_Adjectives set.

From figure 71, we can see that different numbers of starting adjectives do not vary the ranking results much in terms of their percentage of overlap with the box office ranking. Even when we only use 1 starting adjective, our method still works in producing high quality ranking. From k = 6, the percentage of overlap with the box office ranking is always higher than 60% for various number of starting adjectives.

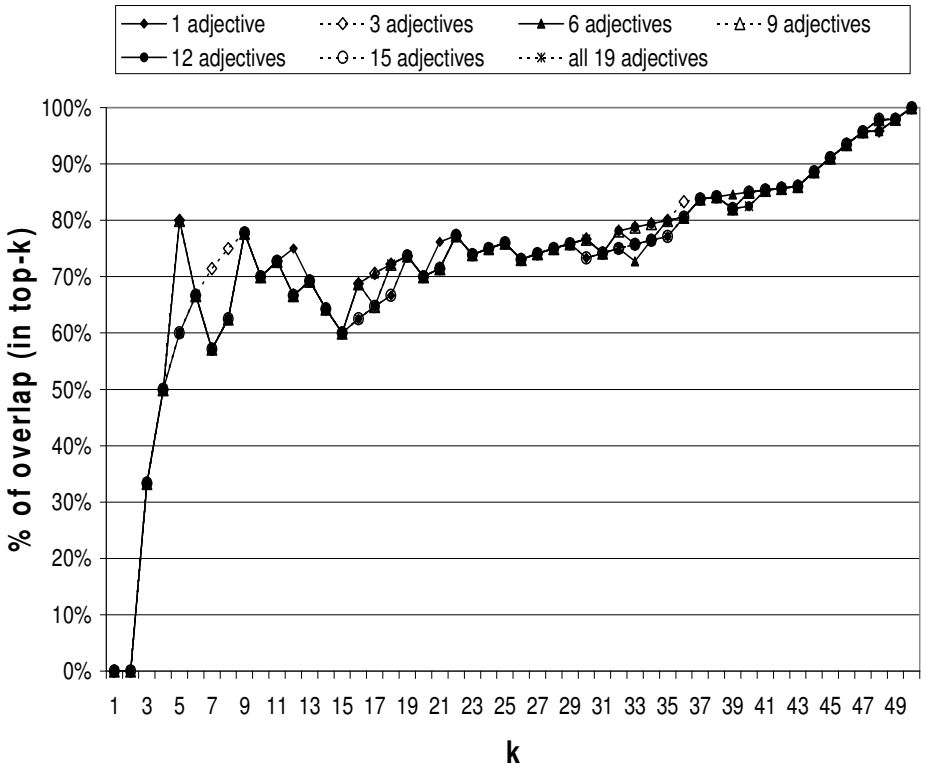

Fig. 71. Percentage of Overlap in top-k Movies (individualPositive method)

In figure 72, we present the results of running individualPositive method with different subsets of starting adjectives extracted from our Good_Adjectives set.

From figure 72, we can see that different subset of starting adjectives do not vary the ranking results much in terms of their percentage of overlap with the box office ranking.

These results show that our method is not overly sensitive to the number or the choice of starting adjectives. The sentiment graph itself may have contained enough information on the semantic orientations of its vertices (adjectives).

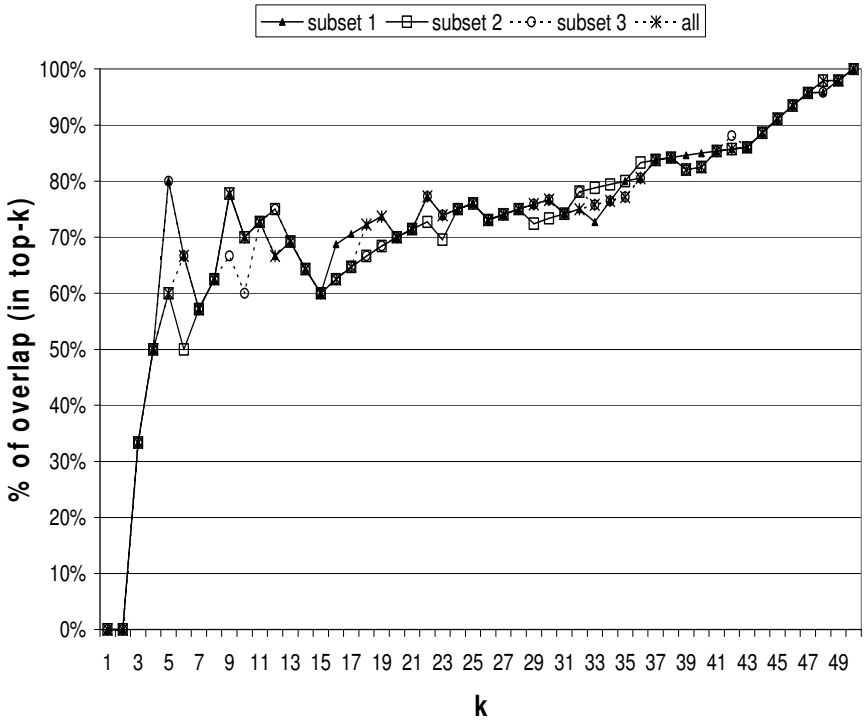

Fig. 72. Percentage of Overlap in top-k Movies (individualPositive method)

5.2.4.4 Ranking of Adjectives

The interesting thing about our proposed methods is that, not only they can produce the ranking of the movies; they can also produce the context-dependent ranking of the adjectives in the reviews. Here we present some interesting results from the ranking of the adjectives when we conduct our method: byGenrePositive using only one positive adjective: "good" in our starting set.

Using only the adjective "good" as its starting adjective, our method is able to find that the adjective "great" has also a universal positive semantic orientation in all the genres: i.e. "great" is ranked in the top 1% of adjectives with highest positive orientations in all the genres.

Among other interesting top 1% adjectives with highest positive orientations are: "funny" (in comedy, chick flick, animation and children genres), "stupid" (in the comedy genre), "animated" (in animation and children genres), "musical" (in the musical genre), "political" and "flawed"

(in the political genre), “original” (in the psycho genre), “enchanted” and “fairy” (in the children genre), “real” (in the drama genre), “young” and “British” (in the romantic genre). Some of these adjectives have either ambiguous orientations or orientations that are genre-specific.

For example, the adjectives “flawed” and “stupid” have ambiguous semantic orientations: i.e. they can have positive or negative semantic orientations depending on how they are used in the sentence. Our method is able to identify that these adjectives have positive orientations in the political and comedy genre, respectively. Further investigation to the actual reviews reveals interesting usage of the adjective “flawed” in the political genre: “… a rather affectionate look at a flawed man who felt compelled to right what was wrong”, “Wilson Hanks, a flawed and fun loving Congressman from the piney woods of East Texas…”, and the interesting usage of the adjective “stupid” in the comedy genre: “I like a stupid movie where I do not have to think in and just sit back”, which suggest the positive orientations of the adjective “flawed” and “stupid” in the sentences.

Further, “political”, “musical”, and “animated” are adjectives whose usage and orientations maybe specific only to the political, musical and animation genre respectively. Our method is able to identify that these adjectives have indeed positive orientations in their respective genres even when these adjectives are not in our starting set of positive adjectives.

Lastly, another interesting issue to explore is whether or not the adjectives actually reflect the audience demands for what will be considered good movies for a particular genre. For example, among the top 1% of adjectives (with highest positive semantic orientations) in the romantic genre is the adjective “British”. Indeed, British romantic movie has done continuously well in topping the box office list with movies such as “Bridget Jones’ Diary” and “Four Weddings and a Funeral”. Another example is the adjective “animated” that is ranked among the top 1% of adjectives with highest positive semantic orientations in the children genre. Indeed, animated

movies in children genre have done very well in the past with movies such as “Shrek”, “Finding Nemo”, and “Toys Story”. We are interested in exploring this issue further in future.

5.2.5 Time Series Correlation

Here we present results of experiments to measure the performance of our proposed method that computes correlation between time series as similarity between their corresponding gradient sequences. Two time series are positively correlated if their corresponding gradient sequences have high similarity. We compare the performance of our method with the Longest Common Subsequence (LCS) algorithm in identifying pairs of time series with high similarity in their gradient sequences. We conduct the experiments on both synthetic and real data sets.

In the experiments, we define our kernel function as:

$$K(m_i) = \frac{|p| + |q|}{2 * (1 + 2|p - q|^2)} \qquad (28)$$

where m_i is a mapping from character p with value |p| to character q with value |q|. The character values represent gradient values that are discretized using PAA technique [69]. The mapping that has smaller difference in character values will get a higher kernel value.

5.2.5.1 Synthetic Data Set

In the synthetic data set, each time series (time series 0 to 6) has an obvious pattern of change (cf. figure 73). The more similar the patterns between two time series, the higher the similarity between their gradient sequences, and the more positively correlated they are. In figure 73, we can see that time series 1 is most positively correlated with time series 2, time series 3 is most positively correlated with time series 4, and time series 5 is most positively correlated with time series 6.

The correlation measures produced by our method on this synthetic data set are shown in figure 74 while the correlation measures produced by LCS are shown in figure 75.

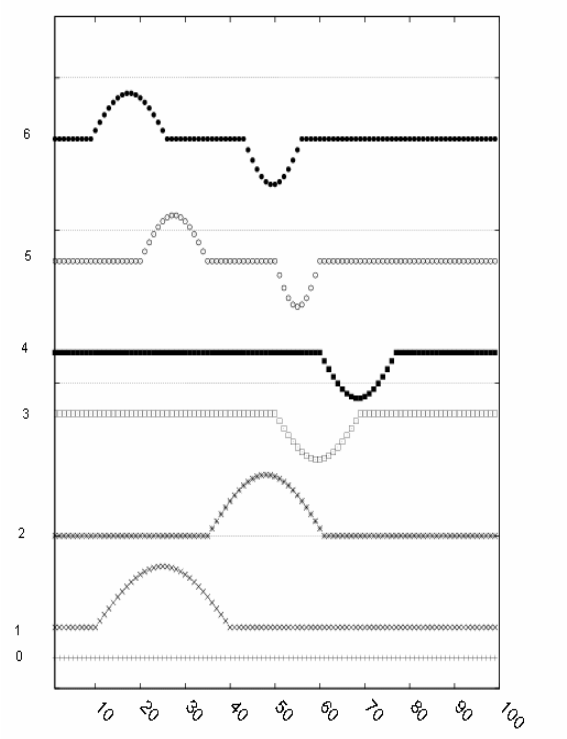

Fig. 73. Synthetic data sets

From figure 74, we can see that our method is able to identify the correct match for each time series: i.e. the time series with highest positive correlation. The correlation measures produced by our method clearly distinguish pairs with high positive correlation from other pairs.

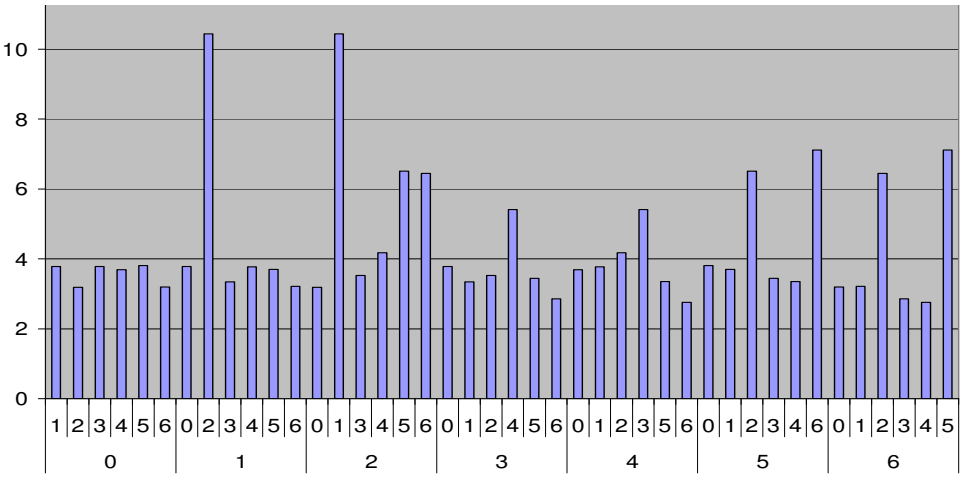

Fig. 74. Correlation Measures Produced by Our Method

From figure 75, we can see that, unlike our method, LCS is not able to identify the correct match for each time series.

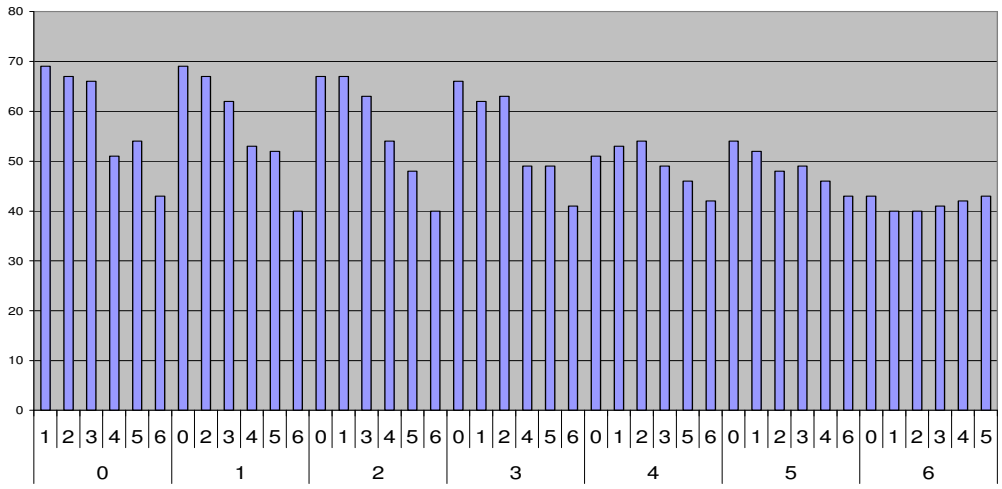

Fig. 75. Correlation Measures Produced by LCS

5.2.5.2 Real Data Set

We also conduct experiment on real data sets. We randomly select the stock price time series of some IT companies including Microsoft (MSFT), EBay (EBAY), Sun (SUN) and Oracle (ORCL) in year 2000 (cf. figure 76).

Based on news identified in [72], in the year 2000 in terms of business, Microsoft is closer to EBay and Sun is closer to Oracle. Therefore, some industry factors may affect their stock price to respond in a similar way. Our goal is to mine this correlation.

Correlation measures produced by our method on this real data set are shown in Figure 77.

From figure 77, we can see that our method has successfully mined the correlation between the stock prices of these companies. Our method is able to identify that the stock price of Microsoft has highest correlation with that of EBay. Our method is also able to identify that the stock price of Sun has highest correlation with that of Oracle

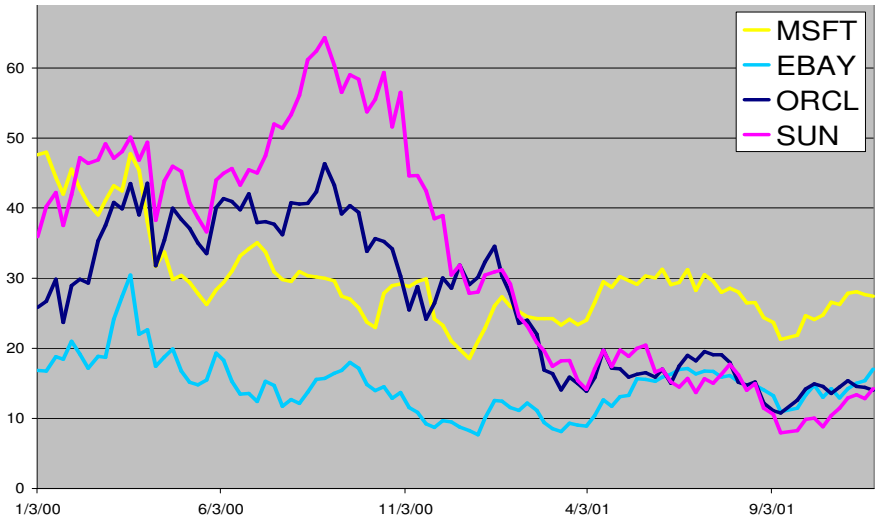

Fig. 76. Real Data Set for IT companies

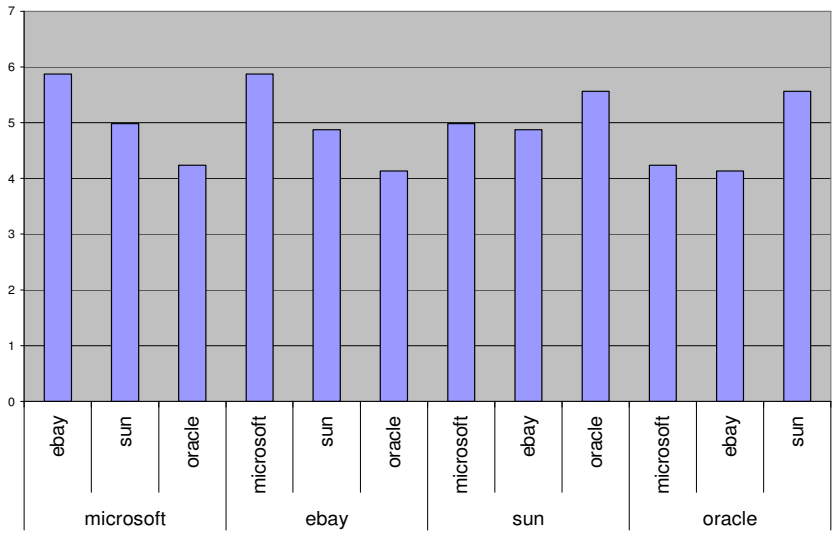

Fig. 77. Correlation Measures Produced by Our method

CHAPTER 6. CONCLUSION

Here we summarize our contributions on the study of graphs, clustering and applications. We also present ideas for future works.

Section 6.1 Graphs

6.1.1 Graph Algorithms

6.1.1.1 Random Walk

We have proposed two novel methods that are based on random walk to rank objects according to opinions in their reviews and to measure correlation between time series. The details of our contributions are presented in the applications section 6.2.4 and 6.2.5 respectively.

6.1.1.2 A Variant of Randomized MST Algorithm

From our study of the randomized MST algorithm, we identify an interesting question with regards to the application of Borůvka steps in the algorithm. The number of Borůvka steps to apply at each recursive call of the algorithm is not clear – it can be 2, 4, etc.

Furthermore, based on the cycle property of graphs, any random tree in the graph can in fact be used to identify *heavy* edges in the graph: i.e. edges that will not be in the MST of the graph. Yet, the randomized MST algorithm, instead of simply using any random tree in the graph, extracts a random subgraph and finds the MST of this subgraph before using the MST to identify *heavy* edges.

We propose a variant of randomized MST algorithm that does not use Borůvka steps and selects any random tree from the graph to identify *heavy* edges. The algorithm incrementally finds the MST of the graph by building on the initial random tree. The algorithm is iterative and it is simpler in implementation, based purely on the cycle and cut property of graphs. However, the run time of the algorithm is O (MN), where M is the number of edges in the graph and N is the number of vertices in the graph. This is slower than the running time Θ (M) of the original

randomized MST algorithm. More random algorithms or heuristics can be explored in future to reduce the run time of our proposed algorithm. For example, the algorithm may be able to run faster if the initial random tree selected is as close as possible to the desired MST of the graph.

6.1.2 Graph-based Clustering Algorithms

In graph-based clustering algorithms, we have contributed novel variants of Star clustering algorithm that explore different techniques, including random walk algorithm, to choose centroids (Star centers). Our variants of Star clustering are more effective than, and as efficient as, the original and extended Star clustering algorithms.

Based on our study of minimum spanning tree algorithm, we propose a novel family of clustering algorithms: MST-Sim that uses minimum spanning tree algorithm and adds to it intra- and inter-cluster similarity metrics, which have basis in graph theory. MST-Sim algorithms are generally very fast and efficient in comparison with other state of the art graph-based clustering algorithms. One of them, MST-Sim Diameter, at optimal setting of the fine tuning parameter, is more effective than all the other clustering algorithms that we have evaluated.

Based on our study of Star clustering, minimum spanning tree and K-means, we propose an unconstrained family of graph-based clustering algorithms named Ricochet that does not require any parameter to be defined a priori. While the fact that Ricochet is unconstrained is already an advantage, Ricochet algorithms are competitive to other state of the art clustering algorithms. One of them, Ordered Concurrent Rippling, yields a very respectable effectiveness while being efficient.

Among our three proposed families of algorithms – variant of Star, MST-Sim, and Ricochet – MST-Sim is the most effective and efficient.

6.1.2.1 Star Clustering

We suspected that the metrics used for selecting star centers in Star clustering is not optimal. The theoretical argument was presented in the original papers of Star clustering but was not exploited. We have therefore proposed various new metrics for selecting star centers: Markov metrics that find vertices of maximum flow; and metrics estimating and commensurating to the maximum intra-cluster similarity (lower bound, average and sum). We empirically studied the performance of off-line star, extended restricted and unrestricted Star and on-line Star with these different metrics.

Our results confirm our conjecture: selecting star centers based on degree (as proposed by the original algorithm inventors) performs almost as poorly as a random selection. One needs to use a metrics that maximizes intra-cluster similarity such as the lower bound metrics. While it indeed yields the best results, it is expensive to compute. The average metrics is a fast and good approximation of the lower bound metrics in all variants of Star algorithm: (1) Star-ave yields up to 4.53% improvement of F1 with a 19.1% improvement on precision and 1.77% on recall; (2) Star-extended-ave-(r) yields up to 14.65% improvement of F1 with a 27.2% improvement on precision and 0.25% on recall; (3) Star-extended-ave-(u) yields up to 138% improvement of F1 with a 102% improvement on precision and 3.4% on recall; (4) Star-online-ave yields up to an outstanding 20.81% improvement of F1 with a 20.87% improvement on precision and 20.67% on recall. We notice that, since intra-cluster similarity is maximized, it is precision that is mostly improved. We can therefore propose Star-online-ave as a very efficient and very effective graph-based clustering algorithm.

6.1.2.2 MST-Sim Clustering

We have proposed a family of one stage agglomerative MST algorithms that modify either Kruskal's or Borůvka's algorithms to incorporate the progressive creation of clusters using inter- and intra-cluster similarity metrics with basis on graph theory. Our proposed graph-based clustering algorithms are generally very fast and efficient in comparison with other graph-based

clustering algorithms. One of them (MST-Sim Diameter) at optimal setting of the fine tuning parameter, is more effective than all the other clustering algorithms that we have evaluated. Our design was guided by the requirement to create algorithms that can be easily adapted to related problems of local and dynamic nature such as k-member clustering. However, the pre-processing time to construct the graph and compute all pair-wise similarities remains a bottleneck for graph-based clustering algorithms, especially when compared to non graph-based clustering such as K-means and k-member greedy clustering.

6.1.2.3 Ricochet: a Family of Unconstrained Graph-based Clustering

We have also proposed a family of algorithms for the clustering of weighted graphs. Unlike state-of-the-art K-means and Star, the algorithms do not require the a priori setting of extrinsic parameters. Unlike state-of-the-art MCL, they do not require the a priori setting of intrinsic fine tuning parameters. We call them unconstrained.

Ricochet uses the results of our study on Star clustering to select potential centroids. It uses the results of our study on MST algorithm to select edges to maximize intra-cluster similarities. Ricochet also uses the idea of iterative reassignment of vertices to centroids in K-means to determine its terminating condition. It is through the combination of these ideas that Ricochet is unconstrained and able to maximize intra-cluster similarities.

The algorithms have been devised by spinning the metaphor of ripples created by the throwing of stones in a pond. Clusters' centroids are stones and rippling is the iterative assignment of objects to clusters. For the sake of completeness, we have proposed both sequential (in which centroids are chosen one by one) and concurrent (in which every vertex is initially a centroid) versions of the algorithms and variants.

After a comprehensive comparative performance analysis with reference data sets in the domain of document corpora clustering, we conclude that, while all our algorithms are competitive, one

of them, Ordered Concurrent Rippling, yield a very respectable effectiveness while being efficient. However, the pre-processing time remains a bottleneck for our proposed graph-based algorithms when compared to non graph-based algorithms like K-means.

Section 6.2 Applications

6.2.1 Document Clustering

We have illustrated and evaluated the performance of our graph-based clustering algorithms for the task of off-line and on-line document clustering. Our contribution includes the comprehensive performance analysis of the algorithms and their comprehensive comparison to other state of the art clustering algorithms.

6.2.2 k-member Clustering

One of the premises of our work is the opportunity to easily adapt our MST-Sim clustering algorithms to related problems of local and dynamic nature such as k-member clustering. Our contribution includes the proposal of a clustering algorithm: MST-Sim, which is easily adaptable to solve k-member clustering problem by adding the constraint that each cluster must have at least k members.

We have successfully created MST-Sim which is comparably effective (in terms of tightness of the clusters) to the existing algorithms. We also contribute in our comprehensive performance evaluation and comparison of MST-Sim with the k-member greedy clustering algorithm [32] that has been shown to outperform other methods for k-anonymization: Median Partitioning and K-Nearest Neighbor algorithms, on the Adult dataset [73] which is considered a benchmark for evaluating the performance of k-anonymity algorithms.

6.2.3 Clustering for POS-Tagging of Bahasa Indonesia

Our contribution is in the presentation of a tool for the interactive and constrained exploration of part-of-speech classes using structural features. The tool relies on incremental hierarchical

clustering algorithms. Our preliminary results for a corpus in the Indonesian language show that the performance is satisfactory even for a small set of words. We have also observed interesting results that clustering at fine granularity displays semantic significance beyond parts-of-speech tagging, in particular with respect to named-entity and word sense recognition.

6.2.4 Opinion-based Ranking

Our contributions include a novel and practical procedure for ranking items based on opinions in their reviews, a comprehensive performance analysis and a methodology for performance evaluation that includes metrics novel in the ranking domain.

We propose a novel and practical context-dependent ranking procedure that can rank items directly from the text of their reviews. The method uses simple contextual relationships such as collocation, negative collocation and coordination by pivot words such as conjunctions and adverbs to construct a sentiment graph. From a small starting set of adjectives whose orientation is universally known, the orientation of other adjectives in the reviews and the items reviewed can be computed and ranked using the PageRank algorithm. The method has several variants whether it uses positive or negative starting adjectives and whether the sentiment graph is constructed for individual items, by genre, or globally. From the ranking perspective, we have contributed by making it possible to rank items using PageRank applied to a different graph other than the graph where the vertices are the items to rank. We use instead a related graph constructed from the smaller components (terms, adjectives) that express opinions about the items. From the opinion mining perspective, we contribute by using PageRank algorithm to design context-dependent ranking of items based on opinions in their reviews.

We instantiate and evaluate our procedure and its variants for ranking movies using reviews of box office movies. While the design of effective context dependent methods is generally considered a hard topic in natural language processing [62], we show that our method is very

effective and produces a ranking comparable to the one of the box office. Our best performing method uses positive starting adjectives and a sentiment graph constructed for individual items (individualPositive). This interestingly happens to be the simplest method. We also show that our method is not overly sensitive to the number or choice of starting adjectives. We compare our method and the box office ranking with the ranking induced from user ratings and show, if it was necessary, the limitation of the latter. This highlights the practical application of our proposal.

Our performance evaluation examines the usage of several metrics for measuring ranking used by different authors. We show how the notion of information loss can be used in evaluating coarseness when measuring ranking effectiveness at different granularities. This is a novel metric in evaluating ranking. From the experiments, we find that ranking based on negative semantic orientation scores does not give as good a performance as ranking based on positive semantic orientation scores. This highlights the question on how best we should perceive the negative orientation scores. We find that positive and negative orientation may not combine well in a linear fashion and that non-negative orientation may not always mean positive orientation. In future, more combination of positive and negative orientation can be explored.

Naturally, our method can be straightforwardly applied to other items and reviews such as hotels, books and so on. Although we do not report these results here, we have conducted further experiments in other such domains that confirm the general effectiveness of our proposed method. With such automated ranking of items based on reviews, companies could track public response to their products or services, politicians could track public opinions, and so on. In future, we will explore the application of our methods to many more domains.

6.2.5 Time Series Correlation

We have proposed a new definition of time series correlation as the similarity of their gradient sequences. We also put forward a novel fuzzy matching method which provides greater

flexibility to handle the similarity search between time series. Our method combines techniques from machine translation and random walk on graphs. It allows for time shifting and mismatching and is able to find partial patterns. Our experiments show that our method successfully finds correlated time series in both synthetic and real data sets and performs better than the traditional Longest Common Subsequence (LCS) algorithm.

Section 6.3 Future Works

Our contribution is the study of graphs, clustering and applications. We have studied graph algorithms such as random walk and minimum spanning tree algorithm, graph-based clustering algorithms, and their applications.

Based on our study, we have proposed a novel variant of randomized MST algorithm, novel variants of Star clustering algorithm, and two novel graph-based clustering algorithms: MST-Sim and Ricochet. MST-Sim introduces inter- and intra-cluster similarity metrics with basis in graph theory. Ricochet is unconstrained – it does not require any parameter to be defined a priori.

We illustrate and evaluate our proposed graph-based clustering algorithms for the task of off-line and on-line document clustering. We have also successfully created a clustering algorithm for solving k-member clustering and propose to use hierarchical clustering for part-of-speech tagging of Bahasa Indonesia. We have also proposed two novel methods that are based on random walk to rank objects based on opinions in their reviews and to measure time series correlation. We evaluate our proposed methods empirically and provide theoretical analysis on the performance.

In future, further improvements to our proposed methods and applications to other domains can be explored.

For example, in terms of the pre-processing time for graph-based clustering algorithms, methods such as indexing, stream processing, bootstrapping can be explored to reduce the number of pair wise similarities that we need to pre-compute. In terms of graph-based clustering algorithms, it

will also be interesting if we can use randomized graph algorithms such as the one proposed in [19] to do clustering. Interesting relationship between matrix factorizations and clustering can also be explored [81, 82]. Further, from our study of the cut and cycle property of graphs, it appears to us that the cut property maybe closely related to the notion of inter-cluster density while the cycle property maybe closely related to the notion of intra-cluster sparsity. It will be interesting to explore if the cut and cycle property of graphs can be used to do clustering. Another interesting idea is in using the notion of *heavy* or *light* edges to remove "unnecessary" edges from the graph thus thresholding the graph before conducting any clustering on the graph.

In terms of application domains, it will be interesting to apply our opinion-based ranking to other domains such as social network, or to extend our opinion-based ranking to identify truth or lies in textual reviews.

We have published our works on Star Clustering [51] and clustering for part-of-speech tagging [56]. Our other work in opinion-based ranking is currently under review.

CHAPTER 7. BIBLIOGRAPHY

[1] Jain A.K., Murty M. N., Flynn, P.J., Data Clustering: A Review, ACM Computing Surveys, Vol. 31, No. 3, September 1999.

[2] Zahn C.T., Graph Theoretical Methods for Detecting and Describing Gestalt Clusters. IEEE Transactions on Computers, Vol. C-20, No. 1, January 1971.

[3] Johnson S. C., Hierarchical Clustering Schemes. Psychometrika, 2:241-254, 1967.

[4] Van Dongen S.M., Graph clustering by flow simulation - [S.l.]: [s.n.], 2000 - Tekst. - Proefschrift Universiteit Utrecht, 2000.

[5] Van Rijsbergen C. J., Information Retrieval, Buttersworth, London, second edition. 1989.

[6] Kowalski G., Information Retrieval systems – Theory and Implementation, Kluwer Academic Publishers, 1997.

[7] Cutting D. R., Karger D. R., Pedersen J. O., and Tukey J. W., Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections, SIGIR '92, Pages 318 – 329, 1992.

[8] Zamir O., Etzioni O., Madani O., Karp R. M., Fast and Intuitive Clustering of Web Documents, KDD '97, Pages 287-290, 1997.

[9] Nomoto T., Matsumoto Y., A New Approach to Unsupervised Text Summarization, SIGIR 2001: 26 – 34, 2001.

[10] http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/ (visited last in June 2008)

[11] Brin S. and Lawrence P., The anatomy of a large-scale hypertextual Web search engine, Computer Networks and ISDN Systems, 30(1-7):107–117, 1998.

[12] Kashima H., Tsuda, K., Inokuchi A., Marginalized Kernels between Labeled Graphs, Proceedings of the Twentieth International Conference on machine Learning (ICML-2003), Washington DC, 2003.

[13] Aslam J., Pelekhov K., and Rus D., The Star Clustering Algorithm. In Journal of Graph Algorithms and Applications, 8(1) 95–129, 2004.

[14] Random Walks on Graphs, http://www.math.uah.edu/stat/Markov/WalkGraph.xhtml (visited last in June 2008).

[15] M. Gondran, M. Minoux, Graphs and Algorithms, John Wiley and Sons, 1984.

[16] Kruskal J. B., On the shortest spanning subtree and the traveling salesman problem. In Proceedings of the American Mathematical Society. 7, pp. 48–50, 1956.

[17] Boruvka O., "O jistém problému minimálním (About a certain minimal problem)", Práce mor. prírodoved. spol. v Brne III, pp. 3: 37–58, 1926.

[18] Skiena S. S., The Algorithm Design Manual, Telos/Springer-Verlag, New York, 1998.

[19] Karger D.R., Klein P.N., and Tarjan R.E., A Randomized Linear-Time Algorithm to Find Minimum Spanning Trees, Journal of the ACM, Vol. 42, pp. 2:321-328, 1995.

[20] MacQueen J. B., Some Methods for classification and Analysis of Multivariate Observations. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. Berkeley, University of California Press, 1:281-297, 1967.

[21] Kaufman L., Rousseeuw P., Finding groups in data: an introduction to cluster analysis. John Wiley and Sons, New York (1990).

[22] Algorithm 479: A Minimal Spanning Tree Clustering Method [Z]. Communications of the ACM, Vol. 17, Issue 6, Pages: 321 – 323, June 1974.

[23] Zheng X., He P., Tian M., and Yuan F., Algorithm of Documents Clustering based on Minimum Spanning Tree, Proceedings of the Second International Conference on Machine Learning and Cybernetics, Xi'an, 2 – 5 November 2003.

[24] Grygorash O., Zhou Y., and Jorgensen Z., Minimum Spanning Tree Based Clustering Algorithms, 18th IEEE International Conference on Tools with Artificial Intelligence, 2006.

[25] He Y. and Chen L., A threshold criterion, auto-detection and its use in MST-based clustering, Intelligent Data Analysis, Volume 9 Issue 3, pp. 253-271, May 2005.

[26] Karypis G., Han E., Kumar V., CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling. IEEE Computer Vol. 32 No. 8, 68-75, 1999.

[27] Zhao Y., Karypis G., Hierarchical Clustering Algorithms for Document Datasets. Data Mining and Knowledge Discovery Vol. 10, No. 2, 141-168, 2005.

[28] Nieland H., Fast Graph Clustering Algorithm by Flow Simulation. Research and Development ERCIM News No. 42 - July 2000.

[29] Croft W. B., Clustering large files of documents using the single-link method. Journal of the American Society for Information Science, pp 189-195, November 1977.

[30] Voorhees E., The cluster hypothesis revisited. In Proceedings of the 8th SIGIR, pp 95-104, 1985.

[31] García R.J. Gil C., Badía J.M., Porrata A. P., Extended Star Clustering Algorithm. Proc. Of CIARP'03, LNCS 2905, 480-487, 2003.

[32] Byun J., Kamra A., Bertino E., and Li N., Efficient k-anonymity Using Clustering Techniques, Proceedings of International Conference on Database Systems for Advanced Applications (DASFAA), 2007.

[33] Aggarwal G., Feder T., Kenthapadi K., Motwani R., Panigrahy R., Thomas D., and Zhu A., Anonymizing tables, In Proceedings of the Tenth International Conference on Database Theory (ICDT), pp. 246–258, 2005.

[34] Meyerson A. and Williams R., On the complexity of optimal k-anonymity, In the twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), 2004.

[35] Charniak E., Hendrickson C., Jacobson N., and Perkowitz M., Equation for Part-of-speech Tagging, In proceedings of the Eleventh National Conference on Artificial Intelligence: 784 – 789, 1993.

[36] Brill E., Automatic Grammar Induction and Parsing Free Text: A Transformation-based Approach, In Proceedings of ACL 31, Columbus, OH, 1993.

[37] Cutting D., Kupiec J., Pedersen J., and Sibun P., A Practical Part-of-speech Tagger, In the 3rd Conference on Applied Natural Language Processing, Trento, Italy, 1991.

[38] Schutze H., Distributional Part-of-speech Tagging, In EACL7: 141-148, 1999.

[39] Bressan S., Indrajaja L., Part-of-speech Tagging without Training, In Proceedings of IFIP International Conference, INTELLCOMM 2004, Bangkok, Thailand, 2004.

[40] Hatzivassiloglou V. and McKeown K.R., Predicting the semantic orientation of adjectives,, Proceeding of the 35th Annual Meeting of the ACL and the 8th Conference of the European Chapter of the ACL, 1997.

[41] Esuli A. and Sebastiani F., SentiWordNet: A Publicly Available Lexical Resource for Opinion Mining, In Proceedings of LREC-06, 5th Conference on Language Resources and Evaluation, 2006.

[42] Esuli A. and Sebastiani F., PageRanking WordNet synsets: An application to opinion mining, in Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL 2007) Prague, CZ, 2007.

[43] Turney P.D., Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews, Proceedings of the 40th ACL, 2002.

[44] Ding X. and Liu B., the Utility of Linguistic Rules in Opinion Mining, SIGIR, 2007.

[45] Berndt D.J. and Clifford J., Using dynamic time warping to find patterns in time series, In KDD Workshop, pages 359-370, 1994.

[46] Chiu, B., Keogh, E. and Lonardi, S., Probabilistic Discovery of Time Series Motifs, in the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 493-498, 2003.

[47] Keogh, E. and Kasetty, S., On the need for time series data mining benchmarks: A survey and empirical demonstration, In Proc. of SIGKDD, pp 102-111, 2002.

[48] Tung A. K. H., Zhang R., Koudas N., and Ooi, B.C., Similarity search: a matching based approach, in VLDB '06: Proceedings of the 32nd international conference on Very large data base, pages 631-642. VLDB Endowment, 2006.

[49] Motwani R. and Raghavan P., Randomized Algorithms, Cambridge University Press, 1995.

[50] King V., A Simpler Minimum Spanning Tree Verification Algorithm, Workshop on Algorithms and Data Structures, 1995.

[51] Wijaya D., Bressan S., Journey to the Centre of the Star: Various Ways of Finding Star Centers in Star Clustering, DEXA 2007.

[52] Kleinberg J., Tardos E., Algorithm Design, Pearson Education Inc., New York, 2006.

[53] Kalaba R., Graph theory and automatic control, Applied Combinatorial Mathematics, Wiley, New York, 1964.

[54] http://people.hofstra.edu/geotrans/eng/ch2en/meth2en/ch2m2en.html (visited last in December 2006).

[55] Wijaya D., Bressan S., Ricochet: A Family of Unconstrained Algorithms for Graph Clustering, The National University of Singapore School of Computing Technical Report, July 2007.

[56] Wijaya D., Bressan S., Hierarchical Clustering for POS Tagging of the Indonesian Language, accepted for SNLP 2007.

[57] Gil D., On the position of Riau Indonesian in a typology of "Flexible-Syntactic-Category" languages, Oral communication in Workshop on Languages with Flexible Parts-of-Speech Systems, University of Amsterdam. (http://www.fgw.uva.nl/aclc), 2007.

[58] Dave K., Lawrence S., and Pennock D.M., Mining the peanut gallery: Opinion extraction and semantic classification of product reviews, WWW, 2003.

[59] Ghose A., Ipeirotis P. G., and Sundararajan A., Opinion mining using econometrics: A case study on reputation systems, In Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics (ACL 2007), 2007.

[60] Hu M. and Liu B., Mining Opinion Features in Customer Reviews, AAAI-2004, 2004.

[61] Whitelaw C., Garg N., and Argamon S., Using appraisal taxonomies for sentiment analysis, in Proc. Second Midwest Computational Linguistic Colloquium (MCLC), 2005.

[62] Liu B., Hu M. and Cheng J., Opinion Observer: Analyzing and Comparing Opinions on the Web, Proceedings of the 14th international World Wide Web conference (WWW-2005), Chiba, Japan, May 10-14, 2005.

[63] Wiebe J., Wilson T., and Bell M., Identifying collocations for recognizing opinions, In Proceedings of ACL/EACL 2001 Workshop on Collocation, 2001

[64] Wilson T., Wiebe J., and Hoffmann P., Recognizing contextual polarity in phrase-level sentiment analysis, In HLT/EMNLP 2005, 2005.

[65] Brill E., Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging, Computational Linguistics, 1995.

[66] Reynar J. and Ratnaparkhi A., a Maximum Entropy Approach to Identifying Sentence Boundaries, ANLP 1997: 16-19, 1997.

[67] Zhou X. and Pu P., Visual and Multimedia Information Management. Springer, 2002.

[68] Wijaya D., Zhang D., Chong F.W., Mining Correlation among Time Series Using Maximum Fuzzy Alignment, Project for the module CS6220 Advanced Topics in Data Mining, School of Computing, National University of Singapore, November 2007.

[69] Keogh E., Lin J., and Fu A., Hot sax: Efficiently finding the most unusual time series subsequence, In ICDM '05: Proceedings of Fifth IEEE International Conference on Data Mining, pages 226-233, Washington, DC, USA, 2005.

[70] http://www.daviddlewis.com/resources/testcollections/reuters21578/ (visited last in June 2008)

[71] http://trec.nist.gov/data.html (visited last in June 2008)

[72] http://news.google.com.sg (visited last in June 2008)

[73] http://www.cs.utoronto.ca/~delve/data/datasets.html (visited last in June 2007).

[74] Iyengar V. S., Transforming data to satisfy privacy constraints, In ACM Conference on Knowledge Discovery and Data mining (SIGKIDD), 2002.

[75] http://www.countryreports.org/ (visited last in June 2007).

[76] LeFevre K., DeWitt D., and Ramakrishnan R., Incognito: Efficient full-domain k-anonymity, ACM International Conference on Management of Data (SIGMOD), 2005.

[77] Jelita Asian, Williams H., and Tahaghoghi S., A Testbed for Indonesian Text Retrieval, In Proceedings of the 9th Australasian Document Computing Symposium, Melbourne, Australia : 55-58, 2004.

[78] Marcus M., Kim G., Marcinkiewicz M., MacIntyre R., Bies A., Ferguson M., Katz K., and Schasberger B., The Penn Treebank: Annotating Predicate Argument Structure, In ARPA Human Language Technology Workshop, 1994.

[79] Bar-Ilan J., Mat-Hassan M., Levene M., Methods for Comparing Rankings of Search Engine Results, Computer Networks 50 (1448-1463), 2006.

[80] Sneddon J., The Indonesian Language: Its History and Role in Modern Society, USNW Press, 2004.

[81] Ding C., He X., K-means clustering via principal component analysis, Proceedings of the twenty-first international conference on Machine learning, 2004.

[82] Ding C., He X., Simon H.D., On the Equivalence of Nonnegative Matrix Factorization and Spectral Clustering, Proc. SIAM International Conference Data Mining, pp. 606-610, 2005.

CHAPTER 8. APPENDIX

List of Words and Tags

Tag	Words
Verb	masuk bernilai beranggotakan berpendapat bertahan mengikuti menghadiri menghadapi mencapai memberikan mewujudkan mencari menilai menyebutkan mengatakan membuktikan mengakui disampaikan ditahan didampingi dikuasai ditinggali dipimpin dihubungi ditanya dinilai diduga
PRP$	lanjutnya tegasnya tambahnya katanya ujarnya
WRP	apa siapa
WRB	kapan bagaimana kenapa mengapa
CD	satu dua tiga empat lima enam
DT	ini itu tersebut setiap beberapa sebuah seorang seekor
CC	sedangkan termasuk karena seperti dimana jadi meski namun tapi lewat melalui berdasarkan yang dan sehingga
JJ	pelan-pelan terbuka merah kuning terakhir ketiga pertama kedua lambat hati-hati cepat ungu biru hitam
PRP	ia dia kamu beliau anda kalian dirinya aku saya mereka kita kami
RB	jangan tidak belum mampu sedikit boleh mungkin bisa perlu mulai cukup agak sangat keras sulit mudah mesti akan ingin semua banyak ada pernah
NN	gol mobil lembar ekor buah orang terdakwa kemenangan nilai tersangka saksi tokoh-tokoh orang-orang kebijakan kejaksaan keputusan pernyataan pejabat pemain pertandingan pertemuan pengadilan penduduk
IN	tentang mengenai di ke oleh pada dari dengan
NNP	padang medan surabaya jakarta ambon italia belanda jerman cina swiss jepang muzadi wahid sidiq mz bisri september desember juni mei april februari januari maret selasa rabu as singapura oktober agustus juli november minggu sabtu jumat senin kamis 1980 1998 2002 2000 2001 hasyim asyawadi abdurahman nur zainuddin cholil timtim australia malaysia