Toward Scalable Hierarchical Clustering and Co-clustering Methods : application to the Cluster Hypothesis in Information Retrieval

Xinyu Wang

N° d'ordre NNT : 2017LYSE2123

THESE de DOCTORAT DE L'UNIVERSITÉ DE LYON

Opérée au sein de

L'UNIVERSITÉ LUMIÈRE LYON 2

École Doctorale : ED 512 Informatique et Mathématiques

Discipline : Informatique

Soutenue publiquement le 29 novembre 2017, par :

Xinyu WANG

Toward Scalable Hierarchical Clustering and Co-clustering Methods:

Application to the Cluster Hypothesis in Information Retrieval

Devant le jury composé de :

Christel VRAIN, Professeure des universités, Université d'Orléans, Présidente

Lynda TAMINE-LECHANI, Professeure des universités, Université Toulouse 3, Rapporteure

Gilbert SAPORTA, Professeur, Conservatoire National des Arts et Métiers, Rapporteur

Marie-Jeanne LESOT, Maître de Conférences HDR, Université Paris 6, Examinatrice

Jérôme DARMONT, Professeur des universités, Université Lumière Lyon 2, Directeur de thèse

Julien AH-PINE, Maître de Conférences, Université Lumière Lyon 2, Co-Directeur de thèse

Contrat de diffusion

(Projet de) Thèse de doctorat de l'Université de Lyon
opéré par
l'Université Lumière Lyon 2

Laboratoire ERIC (EA 3083)
École Doctorale Informatique et Mathématiques (ED 512)

présenté pour obtenir le grade de
Docteur en Informatique

Toward Scalable Hierarchical Clustering and Co-clustering Methods:
Application to the Cluster Hypothesis in Information Retrieval

---

Par: Xinyu Wang

Présentée et soutenue publiquement le 29 novembre 2017, devant un jury composé de :

Gilbert Saporta, Professeur des Université, Conservatoire National des Arts et Métiers	Rapporteur
Lynda Tamine-Lechani, Professeur des Université, Université Paul Sabatier	Rapporteure
Christel Vrain, Professeur des Université, Université d'Orléans	Présidente
Marie-Jeanne Lesot, Maitre de conférences HDR, Université Pierre et Marie Curie	Examinatrice
Julien Ah-Pine, Maitre de conférences, Université Lumière Lyon 2	Co-directeur
Jérôme Darmont, Professeur des Université, Université Lumière Lyon 2	Directeur

Abstract

As a major type of unsupervised machine learning method, clustering has been widely applied in various tasks. Different clustering methods have different characteristics. Hierarchical clustering, for example, is capable to output a binary tree-like structure, which explicitly illustrates the interconnections among data instances. Co-clustering, on the other hand, generates co-clusters, each containing a subset of data instances and a subset of data attributes. Applying clustering on textual data enables to organize input documents and reveal connections among documents. This characteristic is helpful in many cases, for example, in cluster-based Information Retrieval tasks. As the size of available data increases, demand of computing power increases. In response to this demand, many distributed computing platforms are developed. These platforms use the collective computing powers of commodity machines to parallelize data, assign computing tasks and perform computation concurrently.

In this thesis, we first address text clustering tasks by proposing two clustering methods, Sim_AHC and SHCoClust. They respectively represent a similarity-based hierarchical clustering and a similarity-based hierarchical co-clustering. We examine their properties and performances through mathematical deduction, experimental verification and evaluation. Then we apply these methods in testing the cluster hypothesis, which is the fundamental assumption in cluster-based Information Retrieval. In such tests, we apply the optimal cluster search to evaluation the retrieval effectiveness of different clustering methods. We examine the computing efficiency and compare the results of the proposed tests. In order to perform clustering on larger datasets, we select Apache Spark platform and provide distributed implementation of Sim_AHC and of SHCoClust. For distributed Sim_AHC, we present the designed computing procedure, illustrate confronted difficulties and provide possible solutions. And for SHCoClust, we provide a distributed implementation of its core, spectral embedding. In this implementation, we use several datasets that vary in size to examine scalability.

Keywords: hierarchical clustering, co-clustering, Information Retrieval, the cluster hypothesis, distributed computing.

Résume

Comme une méthode d'apprentissage automatique non supervisé, la classification automatique est largement appliquée dans des tâches diverses. Différentes méthodes de la classification ont leurs caractéristiques uniques. La classification hiérarchique, par exemple, est capable de produire une structure binaire en forme d'arbre, appelée dendrogramme, qui illustre explicitement les interconnexions entre les instances de données. Le co-clustering, d'autre part, génère des co-clusters, contenant chacun un sous-ensemble d'instances de données et un sous-ensemble d'attributs de données. L'application de la classification sur les données textuelles permet d'organiser les documents et de révéler les connexions parmi eux. Cette caractéristique est utile dans de nombreux cas, par exemple, dans les tâches de recherche d'informations basées sur la classification. À mesure que la taille des données disponibles augmente, la demande de puissance du calcul augmente. En réponse à cette demande, de nombreuses plates-formes du calcul distribué sont développées. Ces plates-formes utilisent les puissances du calcul collectives des machines, pour couper les données en morceaux, assigner des tâches du calcul et effectuer des calculs simultanément.

Dans cette thèse, nous travaillons sur des données textuelles. Compte tenu d'un corpus de documents, nous adoptons l'hypothèse de «bag-of-words» et applique le modèle vectoriel. Tout d'abord, nous abordons les tâches de la classification en proposant deux méthodes, Sim_AHC et SHCoClust. Ils représentent respectivement un cadre des méthodes de la classification hiérarchique et une méthode du co-clustering hiérarchique, basé sur la proximité. Nous examinons leurs caractéristiques et performances du calcul, grâce de déductions mathématiques, de vérifications expérimentales et d'évaluations. Ensuite, nous appliquons ces méthodes pour tester l'hypothèse du cluster, qui est l'hypothèse fondamentale dans la recherche d'informations basée sur la classification. Dans de tels tests, nous utilisons la recherche du cluster optimale pour évaluer l'efficacité de recherche pour tout les méthodes hiérarchiques unifiées par Sim_AHC et par SHCoClust . Nous aussi examinons l'efficacité du calcul et comparons les résultats. Afin d'effectuer les méthodes proposées sur des ensembles de données plus vastes, nous sélectionnons la plate-forme

d'Apache Spark et fournissons implémentations distribuées de Sim_AHC et de SHCoClust. Pour le Sim_AHC distribué, nous présentons la procédure du calcul, illustrons les difficultés rencontrées et fournissons des solutions possibles. Et pour SHCoClust, nous fournissons une implémentation distribuée de son noyau, l'intégration spectrale. Dans cette implémentation, nous utilisons plusieurs ensembles de données qui varient en taille pour examiner l'échelle du calcul sur un groupe de noeuds.

Mots-clés: classification ascendante hiérarchique, co-clustering, recherche d'informations, l'hypothèse de cluster, calcul distribué.

Acknowledgments

To me, near four years's PhD study on this thesis is like a mountain hike. The mountain peak is insight, but it is never easy to reach. It takes restless effort to keep moving up step by step. Along the exploration, there were obstacles and difficult moments, but what motivated me to go on were the excitement of getting to know the unknown and getting skilled with the unfamiliar. The knowledge and skills that I obtained in my study are like the beautiful mountain view that is only possible to see and to enjoy at a certain attitude. I am thankful that I am not alone in this journey, there are lovely people who guide me, accompany me and support me along this journey. I really appreciate to have them, and I would like to take this opportunity to give my sincere thanks to:

My supervisors Julien and Jérôme. I'd like to thank them for having chosen me for this thesis, for having helped me go through many administrative difficulties in the early phase of my study, for having provided me valuable advises, suggestions and corrections in my research works, for their effort and time in revising my writings and in organizing thesis defense for me.

My friends and colleagues, Pavel, Rado, Jairo, Adrien, Ciprian, Mr. Chauchat, Julien.C, Philips, Somayeh and other lovely people in ERIC lab. I thank them for their useful comments in my work, for their inspiration of new ideas, for their kindness of offering me financial support when my contract was delayed, for their encouragement, and for offering me opportunities to take my mind off research and to have fun with games, BBQs and beers.

My parents Baozhen and Huibin. I am thankful to have parents like them, who love me with respecting my choices, who understand me and support me with their best. With their love, support and understandings, I could pursue my dreams with least limitations.

My husband Yolan. I am grateful to have him, who accompanies me with his love and his confidence in me.

I also thank the REQUEST project, which financed my PhD study.

Contents

Abstract iii

Résume v

Acknowledgments vii

Contents viii

List of Figures xiii

List of Tables xv

1 Introduction 1
- 1.1 Context . . . 1
- 1.2 The REQUEST Project . . . 2
- 1.3 Challenges . . . 2
- 1.4 Contributions . . . 4
- 1.5 Thesis Outline . . . 7
- 1.6 Notations . . . 7

2 State of the Art 9
- 2.1 Introduction . . . 9
  - 2.1.1 Clustering Texts . . . 10
    - 2.1.1.1 The “Bag-of-Words" Assumption . . . 10
    - 2.1.1.2 The Vector Space Model . . . 10
    - 2.1.1.3 Commonly-used Proximity Measures . . . 11
    - 2.1.1.4 Text Clustering Algorithms: An Overview . . . 11
  - 2.1.2 Distributed and Parallel Computing . . . 13
    - 2.1.2.1 MapReduce . . . 14
    - 2.1.2.2 Apache Spark . . . 17
    - 2.1.2.3 MapReduce v.s. Spark . . . 20
    - 2.1.2.4 Distributed Storage Systems . . . 21
- 2.2 Agglomerative Hierarchical Clustering . . . 22
  - 2.2.1 Overview . . . 22
  - 2.2.2 Conventional Methods and the Lance-Williams Formula . . . 24
    - 2.2.2.1 Conventional Methods . . . 24
    - 2.2.2.2 The Lance-Williams Formula . . . 26
  - 2.2.3 Nearest Neighbor Chain Approaches . . . 26

2.2.4 Approaches for Large Datasets . . . 29
2.2.5 On-line AHC Algorithms . . . 29
2.2.6 Distributed and Parallel Approaches for AHC . . . 30
2.2.6.1 Parallel Approaches . . . 30
2.2.6.2 Distributed Approaches . . . 31
2.3 Co-clustering . . . 31
2.3.1 Overview . . . 31
2.3.2 Latent Block Models . . . 32
2.3.3 Graph Partitioning Approaches . . . 34
2.3.3.1 Graph Laplacians, Properties and Spectral Clustering . . 36
2.3.3.2 Mathematical Insight: Spectral Graph Partitioning by Optimizing *Ncut* . . . 39
2.3.3.3 Co-clustering Documents and Terms Using Spectral Graph Partitioning . . . 41
2.3.4 Co-clustering Using Non-negative Matrix Factorization . . . 45
2.3.4.1 Non-negative Matrix Factorization . . . 45
2.3.4.2 Approaches for Co-clustering . . . 46
2.3.4.3 Connection Between NMF and Spectral Graph Partitioning 47
2.3.5 Distributed and Parallel Approaches . . . 48
2.4 Tests of the Cluster Hypothesis . . . 48
2.4.1 Overview . . . 48
2.4.2 Classic Tests . . . 49
2.4.2.1 Comparison Tests of Cluster-based and Document-based Search . . . 49
2.4.2.2 Tests Using Hierarchical Clustering . . . 50
2.4.3 Refined Tests . . . 52
2.4.4 Language Model-based Tests . . . 53
2.4.5 Applications of the Cluster Hypothesis in IR . . . 54
2.5 Conclusion . . . 55

3 The Similarity-based Agglomerative Hierarchical Clustering Framework 57
3.1 Motivation . . . 57
3.2 The Similarity-based Hierarchical Clustering Framework, Sim_AHC . . . 59
3.2.1 Mathematical Deduction . . . 59
3.2.2 Extension to Kernel Functions . . . 63
3.2.3 Sparsification of the Cosine Similarity Matrix . . . 64
3.3 Experimental Verification . . . 65
3.3.1 Datasets, Preprocessing and Evaluation Measures . . . 65
3.3.2 Experiment Settings and Results . . . 67
3.3.2.1 Equivalence between Sim_AHC and the Lance-Williams Formula . . . 68
3.3.2.2 Impact of Sparsifying Similarities on Scalability . . . 68
3.3.2.3 Impact of Sparsifying Similarities on Clustering Quality . 70
3.4 Discussion . . . 71
3.5 Conclusion . . . 71

4 The Similarity-based Hierarchical Co-clustering Method 73

4.1 Motivation . . . 73
4.2 The Computing Procedure of SHCoClust . . . 74
4.3 Experiments: Clustering Effectiveness and Efficiency . . . 78
4.3.1 Datasets, Preprocessing and Evaluation Measures . . . 78
4.3.2 Comparisons of Clustering Effectiveness . . . 79
4.3.3 Examination of Clustering Efficiency with Sparsification . . . 81
4.3.4 Discussion of Complexity and Scalability . . . 84
4.4 Visualization . . . 85
4.5 Conclusion . . . 86

5 Testing the Cluster Hypothesis 89
5.1 Motivation . . . 89
5.2 Datasets, Preprocessing and Experiment Setting . . . 91
5.3 A New Cluster Hypothesis Test Using Sim_AHC . . . 92
5.3.1 Comparison of Retrieval Effectiveness Among Seven Clustering Methods . . . 93
5.3.2 Influence of Improving Efficiency via Sparsification on Retrieval Effectiveness . . . 98
5.3.3 Summary . . . 101
5.4 A New Cluster Hypothesis Test Using SHCoClust . . . 101
5.4.1 Comparison of Retrieval Effectiveness Among Seven Clustering Methods . . . 102
5.4.2 Impact of Sparsification on Retrieval Effectiveness . . . 107
5.4.3 Summary . . . 108
5.5 Comparison between Two Proposed Tests . . . 110
5.5.1 On Retrieval Effectiveness . . . 110
5.5.2 On Computing Efficiency . . . 113
5.5.3 Summary . . . 113
5.6 Conclusion . . . 113

6 The Distributed Implementations 115
6.1 Introduction . . . 115
6.2 The Distributed Implementation of Sim_AHC . . . 116
6.2.1 Computing Procedure . . . 116
6.2.2 Experiments . . . 118
6.2.2.1 Settings and Configurations . . . 118
6.2.2.2 Spark Web UI . . . 119
6.2.2.3 Exploration and Troubleshooting . . . 121
6.2.3 Summary . . . 127
6.3 The Distributed Implementation of Spectral-embedding . . . 128
6.3.1 Computing Procedure . . . 128
6.3.2 Experiments and Analysis . . . 131
6.4 Conclusion . . . 133

7 Conclusions and Perspectives 135
7.1 Conclusions . . . 135
7.2 Perspectives . . . 137

Bibliography **147**

List of Figures

2.1 Overview of text clustering algorithms . . . 12
2.2 MapReduce execution overview [1] . . . 15
2.3 MapReduce word count example . . . 16
2.4 Growth of RDDs' lineage when counting words in PySpark commands . . 18
2.5 Spark cluster architecture . . . 19
2.6 When MapReduce runs an iterative job . . . 21
2.7 When Spark runs an iterative job . . . 21
2.8 Overview of clustering methods . . . 22
2.9 Example of dendrogram . . . 23
2.10 Commonly used conventional AHC methods [2] . . . 25
2.11 Comparison of conventional AHC methods (left) and the NN-chain algorithm (right) [3] . . . 27
2.12 Dendrograms without and with height inversion . . . 28
2.13 Latent block model as a graphical model . . . 32
2.14 An illustration of (a) a bipartite graph and (b) its partitions. d_i denotes a document, t_j denotes a term, e_{ij} denotes an edge that links d_i and t_j. V_1 and V_2 denote two sub-graphs. . . . 34
2.15 A cockroach graph partitioned by (a) an ideal cut and by (b) a Spectral-SVD method . . . 45

3.1 Results of applying Sim_AHC on the Reuters, SMART and 20NG datasets using linear and Gaussian kernels . . . 69

4.1 Comparisons of clustering quality among conventional AHC, BSC, SHCoClust with and without sparsification . . . 81
4.2 Results of linear kernel and Gaussian kernel with sparsification . . . 83
4.3 Doc-term matrix (a) before and (b) after SHCoClust, (c) dendrogram of co-clusters and (d) content the median-sized co-cluster . . . 85

5.1 Illustration of results in Table 5.3 for each tested dataset . . . 95
5.2 Results of sparsifying S obtained by linear kernel . . . 99
5.3 Results of sparsifying S obtained by Gaussian kernel . . . 100
5.4 Illustration of results in Table 5.6 for each tested dataset . . . 104
5.5 Results of sparsifying S_{co} obtained by linear kernel . . . 108
5.6 Results of sparsifying S_{co} obtained by Gaussian kernel . . . 109

6.1 Computing procedure of distributed Sim_AHC using Spark RDDs. C_x and C_y denotes two clusters, S_{xy} denotes their pairwise similarity and S'_{xy} denotes their self similarity. . . . 117

6.2 Screenshot of Spark web UI homepage 120
6.3 Screenshot of jobs of a Spark application 121
6.4 Screenshot of stages inside a Spark job 121
6.5 Screenshot of DAG of a Spark job 121
6.6 Screenshot of Spark executors 122
6.7 Screenshot of running process using 2cir_10xe5 dataset 125
6.8 Computing procedure of distributed spectral embedding 129

List of Tables

2.1 A few RDDs' transformations and actions functions . . . 17
2.2 Types of commonly-used RDDs . . . 19
2.3 Commonly-used distance measures for numeric and binary data [4] . . . 24
2.4 Graphic and geometric methods for computing $D(C_{ij}, C_k)$ [5] . . . 25
2.5 Lance-Williams formula: methods and parameter values . . . 26

3.1 Descriptions of experimented datasets . . . 66
3.2 Best ARI results for each collection when $\tau = 0$ (baseline) and when $\tau > 0$ (sparsified S) . . . 70

4.1 Experimented Datasets . . . 79
4.2 Highest ARI, relative gain in memory and in time with sparsiciation . . . 84

5.1 Experimented datasets . . . 91
5.2 Clustering methods that obtain at least two lowest averaged E values at $\beta = 0.5, 1$ and 2 . . . 94
5.3 Retrieval effectiveness measured by averaged optimal E values for seven clustering methods in Sim_AHC using linear and Gaussian kernels . . . 96
5.4 Standard deviation of optimal E values for seven clustering methods, corresponding to Table 5.3 . . . 97
5.5 Clustering methods that obtain at least two lowest averaged optimal E values at $\beta = 0.5, 1$ and 2 . . . 103
5.6 Retrieval effectiveness measured by averaged optimal E values for seven clustering methods in SHCoClust using linear and Gaussian kernels . . . 105
5.7 Standard deviation of optimal E values for seven clustering methods, corresponding to Table 5.6 . . . 106
5.8 T values of T-tests with $H_0 : \mu_0 = \mu_1$ and $H_1 : \mu_0 \neq \mu_1$ at $\alpha = 95\%$. μ_0 indicates the mean of optimal E values obtained in Sim_AHC, μ_1 for SHCoClust. Values highlighted in red are smaller than critical value. . . . 112

6.1 Commonly used Spark application properties . . . 119
6.2 Datasets experimented in distributed Sim_AHC . . . 122
6.3 Experimented datasets . . . 131
6.4 Performance of distributed spectral embedding using SMART, AP, WSJ and ZSDF datasets . . . 132

Chapter 1

Introduction

1.1 Context

We are now living in an era of data. We use Twitter to share our opinions, we have Facebook to connect with our families and friends, we view job offers and profiles on LinkedIn, and we search for answers on Google. The Internet, devices and applications are part of our modern life, they make our life more convenient and comfortable. On the other hand, we contribute to provide data through them. Everyday, huge amounts of data are being generated, processed and analyzed. A new concept appeared in news and research articles, “Big Data". It refers to the challenges and technologies that address the four V’s of Big Data, veracity, variety, velocity and volume, which are impossible to be processed with a common method.

Among all types of data (images, logs, texts and videos) that are produced over the Web, textual data is surely one of the types that draws much attention of researchers. Compared to other types of data, textual data is easier to generate, easier to collect and ubiquitous in every domain. Analysis on textual data allows researchers to extract public opinions on Twitter, to identify user groups on Facebook or to recommend jobs for users on LinkedIn. Text clustering is one of the analyses that applies clustering methods on textual data. It provides useful descriptive information on a collection of textual files, and is commonly used in organizing files, e.g., grouping similar documents such as news and tweets.

Information retrieval (IR) is a wide domain that extensively applies textual analysis. The general objective is to retrieve texts, which are supposed to be relevant to a given query, with efficiency and effectiveness. Commonly-used Web search engines such as Google, Bing and Yahoo! are successful IR applications. Such engines are document-based IR

systems, which return a list of ranked documents in response to a user query. Some less famous engines such as Yippy[1] and Carrot2[2] are cluster-based IR systems, which return a list of documents from a number of clusters that are presumably relevant to a user query. To address Big Data, there are challenges exposed to both IR systems in terms of computing efficiency and retrieval effectiveness.

1.2 The REQUEST Project

The REQUEST[3] project, short for REcursive QUEry and Scalable Technologies, is a French national project that finances this thesis. Its general objectives are to explore and develop technologies in the aspects of Big Data analysis, visualization and cloud computing. Seven industrial entities, Thales, SNCF, Talend, Syllabs, Altic, Aldecis Isthma, and six academic laboratories, ERIC (Lyon), LIP6 (Paris), LIMSI (Orsay), LABRI (Bordeaux), L2TI (Paris), UTT (Troyes), initially take part in this project. To address the analytics of massive volume data, REQUEST focuses on the development of scalable approaches based on NoSQL (not only SQL) storage and distributed computing. Three topics are covered in this project:

1. intelligent recursive and iterative IR,
2. scalable algorithms and distributed implementations,
3. new approaches of visualization.

Engaged in the work that covers topic 1 and topic 2, my team emphasizes our research with clustering methods, applications to IR, as well as scalable and distributed algorithms. Our tasks are to find scalable clustering methods, apply them in IR tasks, and implement them in a distributed manner.

1.3 Challenges

There are two types of clustering, flat clustering and hierarchical clustering. The result of a flat clustering method presents clusters without interconnections. There is no information on how clusters and how objects inside a cluster are connected. For example, given an input of six data instances, A, B, C, D, E and F, applying a flat clustering

[1] `https://yippy.com/`
[2] `http://search.carrot2.org/stable/search`
[3] `https://www.thalesgroup.com/fr/worldwide/big-data/news/request-programme-launched`

method may return two clusters, ABC and DEF. But there is no way to know whether A firstly merges with B or with C. And there is no way to know whether D is closer to E or to F. This information, however, is available in hierarchical clustering, which we consider more informative than flat clustering. In IR tasks, this character of hierarchical clustering is advantageous as it organizes a collection of documents with extra details on their connections, and thus it allows an IR system to better guide users in information seek. However, hierarchical clustering is computationally expensive. Its time complexity is $O(N^3)$ for conventional agglomerative methods, and for divisive hierarchical clustering, complexity can be NP-hard. In many cases, hierarchical clustering is not preferred due to its high complexity, despite its advantageous character. It is thus essential to design an innovative method that makes the hierarchical clustering to perform correctly, using limited computing resources. The capability of an algorithm to achieve expected results using limited computing resources (such as memory and computing time) on relatively large dataset is referred to as "scalability". Nevertheless, limited by the computing procedure of the hierarchical clustering, making it scalable is not trivial.

There are many research works that try to improve the efficiency of hierarchical clustering. A common drawback of these works is that they are not generic, i.e., they work on one or a few methods, but are not applicable to other methods of the same type. We are interested in finding a generic framework that is applicable for all conventional hierarchical clustering methods. Another issue found in past works is that they either require some extra structure (for example, a clustering feature tree in the BIRCH method [6]) or deploy a sampling procedure in order to reduce processing time. An obvious disadvantage of sampling is that it is likely to hurt the accuracy of the results. Is it possible to find a generic framework that is scalable and capable to produce deterministic results? This is one of the problems that this thesis devotes to solve for conventional hierarchical clustering.

Another clustering method, co-clustering, is also being widely studied and applied. Unlike usual clustering methods, which only group objects, a co-clustering method groups objects and their features at the same time. It outputs a number of co-clusters, each containing a subset of objects and a subset of features. For example, an individual co-cluster obtained from a collection of documents is composed of a set of similar documents and a set of words that are associated to the set of documents. Co-clusters are like flat clusters. There is no information on how co-clusters and elements inside a co-cluster are connected. As there is not sufficient study on this subject in the past research works, it inspires us to look for a new approach that is capable to simultaneously group objects and features, as well as to retain the connections among elements inside a co-cluster and among co-clusters. The challenge of this task is to find such an approach.

As a branch in the domain of IR, applying clustering methods in IR tasks has been studied for many decades. Cluster-based IR applications have a fundamental assumption, the cluster hypothesis. It states that similar documents in a cluster tends to respond to the same query. There exist many works that test this hypothesis using several conventional hierarchical clustering methods. However, in terms of retrieval effectiveness, these works draw different conclusions. The difference in their conclusions are likely caused by the differences in experimental settings, experimented datasets and evaluation measures. Performing a cluster hypothesis test using a specific clustering method allows us to know how well a query is answered to (retrieval effectiveness) and how fast a query is responded (retrieval efficiency). This knowledge is essential in understanding the performance of a clustering method in an IR task. However, the challenge is: how to perform a test on the cluster hypothesis, given the fact that there are different experimental settings, datasets and evaluation measures.

Distributed computing is becoming more and more popular in processing large datasets. As hardware is commonly available at low cost, it is feasible to group commodity machines and use their collective computing power to handle data processing. With the aid of open source applications that are capable of handling concurrency, job scheduling and data replication, applying distributed computing becomes practical and convenient. Still, there are many technical details to consider in practice in order to achieve efficiency and ensure computing accuracy. And the most challenging part is how to program a user-defined algorithm so that it can be correctly performed on a distributed computing platform.

1.4 Contributions

To serve the REQUEST project's objectives and to tackle the challenges, we introduce the contributions in this thesis.

- We study previous works on hierarchical clustering and co-clustering, analyze their advantages and drawbacks, propose new approaches, verify, experiment and evaluate them. The proposed algorithms are:

    1. Sim_AHC [7], the similarity-based hierarchical clustering framework.
       As mentioned previously, applying AHC is prohibitively expensive due to its high complexity. To address this issue, different AHC algorithms have been proposed in past works (Section 2.2). However, these algorithms are either not generic, or their results are not deterministic due to the use of sampling.

A great interest for us is to find a framework that is generic to all conventional clustering methods and produces deterministic results. Inspired by the connection between similarities and distances, we propose Sim_AHC. It uses similarities instead of distances, and outputs a binary tree-like structure that can thoroughly illustrate the connections of sub-clusters, which are composed of data instances. This is a generic framework for seven conventional hierarchical clustering methods and it outputs deterministic results. More importantly, as its similarities are all between zero and one, a thresholding strategy can be applied to sparsify the similarity matrix in order to achieve computing efficiency.

Through our experiments, we find that the computing efficiency of Sim_AHC can be largely improved in terms of memory use and running time, with clustering quality being guaranteed. In fact, Sim_AHC clusters as well as, or better than conventional AHC methods that use distances. This holds true, even when the similarity matrix is substantially sparsified in Sim_AHC. Besides, thanks to the usage of inner product, any kernel function can be applied in Sim_AHC. These properties make Sim_AHC superior to distance-based hierarchical clustering methods.

2. SHCoClust [8], the similarity-based hierarchical co-clustering method.

   Interested in a hybrid approach that is capable of performing co-clustering while maintaining the connections of elements inside a co-cluster and among co-clusters, we propose SHCoClust. The output is also a binary tree-like structure. However, sub-clusters are composed of both data instances and features. This allows us to explore co-clusters that are organized by a hierarchy. When clustering texts, SHCoClust models a collection of documents as a bipartite graph, whose vertices are documents and terms. The objective is to form several sub-graphs, inside each of which vertices are closely associated. However, the associations among sub-graphs are weak. SHCoClust can be regarded as an extension of Sim_AHC applied in a space that is composed of eigenvectors of the Laplacian matrix of the bipartite graph.

   In terms of clustering quality, our experiments demonstrate that SHCoClust significantly outperforms the conventional hierarchical clustering methods. In comparison to the Spectral Bipartite Co-clustering method [9], improvement in clustering quality is obtained in SHCoClust when the input similarity matrix is sparsified. Besides, SHCoClust inherits the advantageous property of Sim_AHC in attenuating high complexity. As inner product based similarities are used, its input similarity matrix can be sparsified in order to achieve better efficiency, and this strategy does not harm the clustering quality.

- Application to testing the cluster hypothesis.

  We propose two tests on the cluster hypothesis, using Sim_AHC and SHCoClust, respectively. There are several interests for us to perform these tests: firstly, they allow us to understand how well and how fast a query is responded using a clustering method. Secondly, some of past works that test the cluster hypothesis conclude differently on which clustering method has better retrieval effectiveness. We are interested to provide a benchmark on this issue. Besides, we discover that only four conventional clustering methods are tested previously, leaving the conclusions on the other methods unknown. Thus the third interest of our tests is to provide a complete conclusion for all methods. Lastly, retrieval efficiency is barely discussed in these tests. As an important factor that measures performance, it is essential to provide experimental evidence on this issue. In our experiments, we examine the efficiency by addressing the impact of sparsifying the similarity matrix. Besides, comparisons between the two tests are provided in terms of retrieval effectiveness and efficiency.

  In our experiments, optimal cluster Search is applied to search for the optimal cluster for a given query. We use the E-measure to evaluate retrieval effectiveness. Similarities that are generated by linear kernel and Gaussian kernel are used in all experiments. In the test that applies Sim_AHC, we find that the average link and the Ward method are the most effective clustering methods. In contrast, in the test that applies SHCoClust, a wider rage of methods such as centroid, single link and McQuitty demonstrate better effectiveness than the others in some cases. In terms of efficiency, we show that the retrieval effectiveness is barely affected by sparsification in both tests. Concretely, when the similarity matrix is getting more and more sparsified, retrieval effectiveness tends to be invariant.

- Implementation using a recent distributed computing platform. After comparing different distributed computing architectures, we select the Apache Spark Engine[4] since it best suits our needs for computation speed and for compatibility. Utilizing its Resilient Distributed Databases (RDDs), we implement two algorithms, the distributed Sim_AHC and the distributed spectral embedding. Implementing a user-defined program in Spark is not a trivial task. Unlike in conventional programming, the performance of a Spark program is also dependent on the number partitions of RDDs, the length of RDD's lineage, the assigned number of cores, the assigned distributed memory, as well as the choice of caching or uncaching an RDD. There is no rule of thumb on how to choose these parameters, because they are really task-dependent. The only way is to explore along programming and to

[4] `spark.apache.org/`

learn a suitable setting by experimenting. In this thesis, apart from implementation details, we also provide the learned experience in each implementation.

1.5 Thesis Outline

This thesis is composed of seven chapters, they are structure as follows:

In Chapter 2, we present state of the art, in which we detail background knowledge, related concepts, previously proposed approaches, existing techniques and software that concern (1) agglomerative hierarchical clustering, (2) co-clustering, (3) the cluster hypothesis tests and (4) distributed computing and architectures.

Chapter 3 and Chapter 4 respectively illustrate the proposed algorithms, Sim_AHC and SHCoClust. In detail, we exhibit, elaborate on and discuss mathematical deduction, computing procedures, properties, experimental verification, results and visualization. In the experiments, we carry out text clustering tasks on several datasets.

Chapter 5 presents the applications of testing the cluster hypothesis using the proposed methods. Unlike in text clustering, we use the E-measure to evaluate retrieval effectiveness in the context of optimal cluster search, and we adopt different experimental settings. This chapter is subdivided into three parts. In the first two parts, we elaborate on the proposed tests on the cluster hypothesis. In each individual part, we address two subjects: (1) a comparison of retrieval effectiveness among seven conventional hierarchical clustering methods, and (2) the impact of sparsification on retrieval effectiveness and efficiency. The third part of this chapter contributes to a comparison between the two tests on overall retrieval effectiveness and on general computing efficiency.

Chapter 6 shows the details of the distributed implementations. We present data structure, computing settings, technical problems as well as solutions.

Finally, chapter 7 concludes this thesis and exhibits research perspectives.

1.6 Notations

$\mathcal{D}$	a dataset
D	pairwise dissimilarity matrix
n	the number of data instances/samples
m	the number of data attributes/features
i	a row index, $i = 1, \ldots, n$
j	a column index, $j = 1, \ldots, m$
A	a data matrix of n-by-m
K	the number of desired clusters
A	a data matrix of n-by-m
a_{ij}	a value in the i_{th} row and the j_{th} column of A
z	a partition of data instances
w	a partition on data attributes
g	the number of partitions of data instances
h	the number of partitions of data attributes
k	an index of z, $k = 1, \ldots, g$
l	an index of w, $l = 1, \ldots, h$
G	a bipartite graph
V	the set of vertices of G, $v_i \in V$
V_i	a subset of V
E	the set of edges
d	a document
t	a term
e	an edge
W	the adjacency matrix of G
w_{ij}	an element of W
D	the degree matrix
deg_i	degree of vertex v_i
L	the graph Laplacian matrix
v	an eigenvector
V	a matrix with eigenvectors v_i as columns
C_i	a cluster
y_i	the i-th row of V
A_n	the scaled weighted matrix of A
M	the number of nonzero values in A_n
S	a similarity matrix, $S = (s_{ij})_{i,j=1,\ldots,n}$
$\mathbb{F}$	a factorized matrix from A
G	a factorized matrix from A

Chapter 2

State of the Art

2.1 Introduction

In the State of the Art, we present concepts, existing approaches, techniques that concern the content of this thesis in detail. As the principle tasks of this thesis are about clustering texts, so this chapter starts with fundamental knowledge on this subject. This includes the basic assumption, the base model, and commonly-used proximity measures for text clustering. After we present these concepts, an overview of popular and recent text clustering algorithms is presented. Since this thesis also concerns implementations of the proposed methods using distributed computing, we also introduce some fundamental knowledge on distributed and parallel computing. In this part, readers can have a better idea of the techniques chosen in this thesis. After presenting necessary concepts in Section 2.1, we develop three more sections in this chapter. Each is on a specific topic. These topics are the agglomerative hierarchical clustering, the co-clustering, and the tests of the cluster hypothesis.

For the agglomerative hierarchical clustering, we first introduce the conventional methods and the Lance-Williams formula. After this, several approaches of the nearest neighbor chain are illustrated. In the end of this Section, some on-line AHC algorithms, as well as the distributed and parallel approaches of AHC are presented.

The section of co-clustering starts with the approaches of the latent block model, a group of statistical models. Then the graph partitioning approach is presented. As this approach is used in SHCoClust, which is proposed in the thesis, more details of mathematical insight and properties are provided. Furthermore, several existing methods of this approach are elaborated on and compared among. This section also contains some algorithms that apply non-negative matrix factorization in co-clustering. As it is in fact

related to the core method of the graph partitioning method, an explanation on this relation is given in the end of the section.

The last section of this chapter contributes to the tests of the cluster hypothesis. As this thesis proposes two new tests on this hypothesis, it is necessary to present concepts and tests on this subject. Concretely, the classic tests, the refined tests and the language-model-based tests are presented. In the end of this section, applications of the cluster hypothesis in IR is shortly discussed.

The objective of this chapter is to present, in details, the concepts, approaches, comparisons and discussions that are indispensable for readers to understand the content of this thesis.

2.1.1 Clustering Texts

2.1.1.1 The “Bag-of-Words" Assumption

Clustering textual data groups similar documents and reveals hidden connections. As textual data is more complex than numeric data, it requires to be treated differently. There are a few assumptions for processing textual data, different assumptions lead to different approaches. The “bag-of-words" assumption is one of the most popular ones. It considers a piece of text (or a document) as a set of words. In this assumption, the ordering of words is ignored, only their existence matters. There are other assumptions that believe the ordering of words conveys necessary information, which is taken as features in corresponding models.

Depending on the choice of assumption, clustering methods on textual data vary. In general, these methods can be categorized into distance-based algorithms, phrase-based algorithms, probabilistic generative models, textual streams methods and graph approaches. In the scope of this thesis, we focus on the distance-based algorithms and develop our approaches on top of the bag-of-words assumption, so that we can compare our methods with past research works, most of which apply the same setting.

2.1.1.2 The Vector Space Model

Based on the bag-of-words assumption, textual data can be represented in the vector space model, in which any document in a given collection is considered as a vector with its contained unique terms (or words) as features (or attributes). In such a way, a collection of documents can be treated as a matrix, in which each row is a document vector and each column is indexed by a term. Depending on specific needs, this document-term

matrix can either be binary, or filled with term weights. Commonly-used term weighting schemes are term frequency (TF), term frequency-inverse document frequency (TF-IDF) and BM25. In each scheme, a term in the vocabulary is assigned with a weight, reflecting its importance in the collection. As documents often contain different number of terms, the document-term matrix is usually normalized, standardized or scaled. If documents contain too many terms, feature reduction is necessary. It helps reduce the number of features and to transform the features into a different space, where clustering or other learning tasks can be performed with better effectiveness.

2.1.1.3 Commonly-used Proximity Measures

For distance-based algorithms and when the TF-IDF scheme is applied, cosine similarity is often used to measure the proximity between two documents d_i and d_j:

$$cosine(d_i, d_j) = \frac{< d_i \cdot d_j >}{\|d_i\| \cdot \|d_j\|}.$$

where $< \cdot >$ indicates an inner product, and $||d||$ indicates the Euclidean norm of a document vector d. Under the vector space model, where documents are featured by their terms, the cosine similarity measures the angle of two document vectors in the projected space. Being 0 means that the document vectors are orthogonal, thus entirely dissimilar; and being 1 indicates that the document vectors are pointing to the same direction, they are entirely identical.

Depending on the choice of models, different proximity measures should be used. For example, when documents are represented as probability distributions over terms, Kullback-Leibler divergence [10] is often used as a proximity measure. It measures how one probability distribution diverges from a second expected probability distribution.

2.1.1.4 Text Clustering Algorithms: An Overview

Aggarwal and Zhai [10] present two major types of clustering algorithms: distance-based methods and word-phrase-based methods. Distance-based clustering algorithms contain two members, agglomeartive hierarchical clustering (AHC) and distance-based partitioning algorithms (Figure 2.1). The AHC approach processes the pairwise distances of input documents, and outputs a binary tree structure that explicitly presents the interconnections among input documents. More details on this approach are provided in Section 2.2. In contrast, the K-medoid and the K-means algorithms are partitioning algorithms. They return flat clusters of documents. The distances of documents to their cluster representatives are supposed to be minimized in convergence. A hybrid

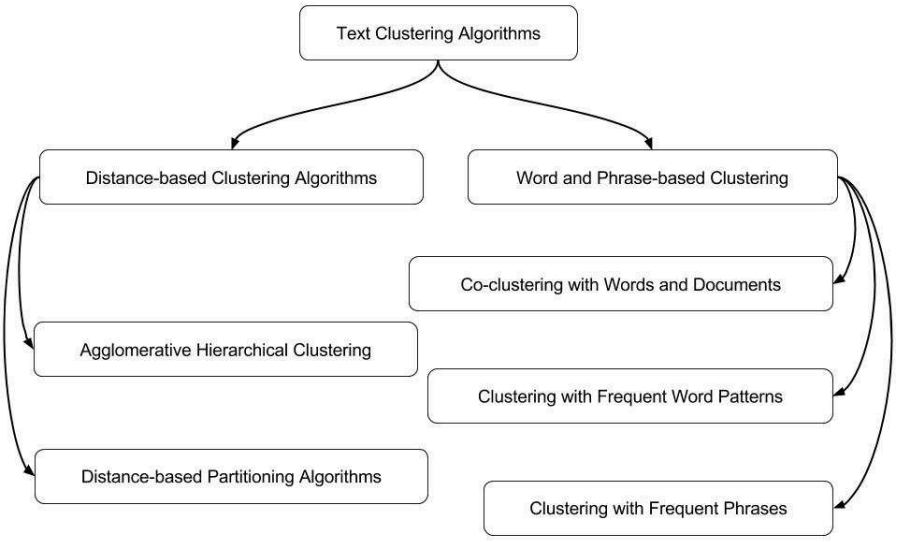


FIGURE 2.1: Overview of text clustering algorithms

approach that uses both hierarchical and partitional algorithms is introduced, it is called the Scatter-Gather clustering method [10, 11]. This method uses a hierarchical clustering algorithm on a sample of the input documents in order to determine a robust initial set of seeds, which are then used in conjunction with a standard K-means clustering algorithm to find clusters.

The words-phrase-based clustering approaches include three main methods: (1) co-clustering with words and documents, (2) clustering with frequent words patterns, and (3) clustering with frequent phrases. Co-clustering is related to subspace clustering. It groups documents and terms simultaneously and outputs co-clusters, each of which contains a partition of documents and a partition of words. Different co-clustering methods are detailed in Section 2.3. As one representative method of clustering with frequent words patterns, the approach introduced in [12] uses frequent terms sets for text clustering. The main idea of this approach considers low dimensional frequent term sets as cluster candidates, i.e., a frequent term set is regarded as a descriptor of a cluster. Looking for a set of carefully chosen frequent term sets is thus a clustering process, which is based on the mutual overlap of frequent sets with respect to the sets of supporting documents. Clustering with frequent phrases differs from other text clustering methods by treating a document as a string instead a bag of words. So that the ordering information of words in a document is retained. This clustering approach uses an indexing method to organize the phrases in the collection of documents, and then uses this organization to create clusters. In [13], the authors introduce Suffix Tree Clustering, which identifies sets of documents that share common phrases and uses this information to create clusters and to summarize their content for users.

There are other text clustering approaches, such as probabilistic document clustering and semi-supervised methods. Topic modeling is an example of probabilistic document clustering. The idea is to create a probabilistic generative model that represents a collection of documents as a function of hidden random variables, whose parameters are estimated from the document collection [10]. PLSI (Probabilistic Latent Semantic Indexing) [14] and LDA (Latent Dirichlet Allocation) [15] are two representative methods for topic modeling. Semi-supervised learning is a technique situated between supervised and non-supervised learning. With some prior but incomplete knowledge of document labels, semi-supervised learning processes such knowledge to perform a clustering or a classification task. There exist different semi-supervised approaches in text clustering. For instance, [16], a method that incorporates supervision in a non-supervised partitional clustering method; [17] represents a number of semi-supervised learning using probabilistic frameworks; and [18] is a graph-based method that incorporates prior knowledge in the clustering process.

In this thesis, we focus on one of the distance-based text clustering methods, agglomerative hierarchical clustering, and one of the word-phrase-based clustering approaches, co-clustering with words and documents.

2.1.2 Distributed and Parallel Computing

A conventional computer program processes a given input sequentially using available resources of a single machine. If the given input is too large to fit in the memory of the machine, techniques that take advantage of concurrent computing are practically required. Bekkerman et al. [19] point out that there are two major directions in which concurrent execution of tasks can be realized: data parallelism and task parallelism. The former refers to executing the same computation on multiple inputs concurrently, and the latter refers to segmenting the overall algorithm into parts, some of which can be executed concurrently. With the development of hardware technologies, computing devices become cheaper and more powerful. This trend offers remarkable increase in computing capabilities to handle both data and task parallelism.

Parallel computing platform and distributed computing platform are the two types of platforms that support data parallelism and task parallelism, respectively. The parallel computing platform is characterized by organizing processing units in some structure, and by the access to shared memory via direct communication among the units. Modern parallel platforms are usually based on hybrid typologies, in which processing units are organized hierarchically, with multiple layers of shared memory [19]. The Graphics Processing Units (GPUs) are a good example of such an architecture. GPUs are usually

composed of dozens of multiprocessors, each is comprised of multiple stream processors. These stream processors are organized in “blocks", and any individual block has access to relatively small locally shared memory and a much larger globally shared memory [20]. Other parallel platforms include multi-core processing and Message Passing Interface (MPI).

Compared to parallel platforms, distributed computing platforms typically have larger physical distances among processing units, for which less direct communication is usually supported by local (or remote) network. Therefore, they have higher latency and lower bandwidth. However, distributed computing platforms have advantages. For example, individual processing units can be heterogeneous. Moreover, these platforms automatically schedule jobs, synchronize, transfer and recover data (data recovery only happens when a computing unit fails to respond). MapReduce, Apache Spark, Dryad, Pregel and CIEL are a few popular distributed platforms. They actually use a distributed storage system to store data, but to a user, all the computation is like running on a single machine. Dean and Ghemawat [1], Zaharia et al. [21], Ingersoll [22], Gropp [23], Barney [24] provide detailed insights on these distributed platforms.

The distributed implementations illustrated in this thesis are built on the Apache Spark platform[1], which is an advanced directed acyclic graph (DAG) execution engine that supports cyclic data flow and in-memory computing. The core concept of Spark are the resilient distributed datasets (RDDs), which makes Spark low I/O cost and fast computing engine. Spark has been shown to be 100 timess faster than Hadoop MapReduce. Moreover, it provides user-friendly APIs (Application Programming Interfaces) and rich Machine Learning libraries for Python, R, Java and Scala. Despite many advantages, Spark is developed from the MapReduce computing scheme.

2.1.2.1 MapReduce

First introduced by J. Dean and S. Ghemawat from Google, MapReduce [1] is a programming model for processing large datasets. The main idea of this model is based on two types of functions: a map function that processes a key-value pair to generate a set of intermediate key-value pairs; and a reduce function that merges all intermediate values associated with the same intermediate keys. A practical convenience provided by this model is that programs written in this functional style are automatically parallelized and executed on a cluster of machines. This advantage exempts programmers from the details of partitioning the input data, scheduling the executing tasks across a set of machines, handling machine failures, and managing inter-machine communication.

[1] `https://spark.apache.org/`

The machines that MapReduce functions operate on are called nodes. One node executes the master program, while the other nodes run copies of a worker program. The master is responsible for assigning tasks to workers and storing the status of assigned task on each worker. During the execution of a job, master and workers interact in both the map and the reduce phases. Dean and Ghemawat [1] summarize seven steps that are involved in executing a MapReduce job (Figure 2.2).

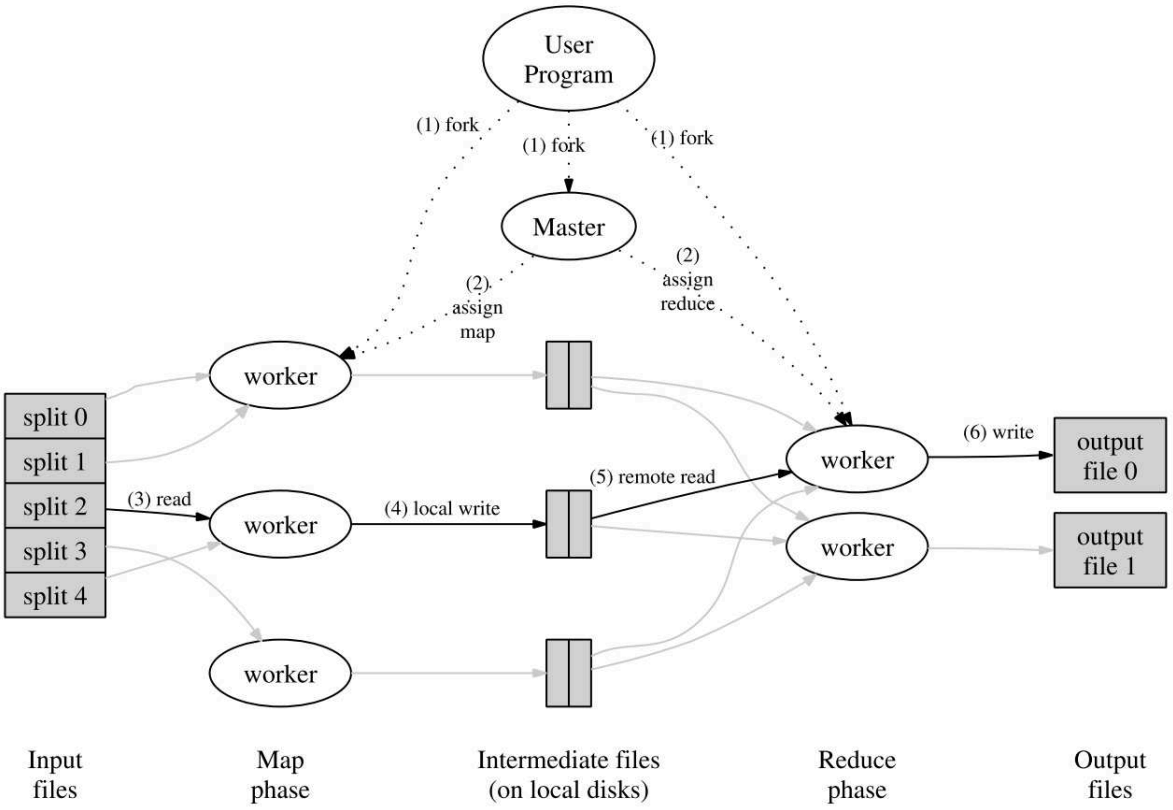


FIGURE 2.2: MapReduce execution overview [1]

1. The input is firstly split into M pieces by the MapReduce library in the user program. Then many copies of the program are copied on nodes.

2. The master program is responsible to assign tasks to workers. There are M map tasks and R reduce tasks to assign. Once an idle worker is detected, a task is then assigned to it.

3. If the worker is assigned a map task, it reads from the corresponding input split, and parses the content into key-value pairs. These pairs are later passed to the user-defined map function, which outputs intermediate key-value pairs and buffers them in memory.

4. The buffered pairs are periodically written to local disk and are partitioned into R regions by a partitioning function. The master is informed of the locations of these buffered pairs on the local disks of the map workers, and it forwards these locations to the reduce workers.

5. Once a reduce worker is notified of these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers. After it finishes reading all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together. The sorting phase is necessary as many different keys map to the same reduce task. If the amount of intermediate data is too large to fit in memory, an external sort is used.

6. The reduce worker iterates over the sorted intermediate data. Once a unique intermediate key is encountered, the reduce worker passes this intermediate key and its values to the user's reduce function. The output of the reduce function is appended to a final output file for this reduce partition.

7. Once all map and reduce tasks are completed, the master wakes up the user program, and the user program returns back to the user code.

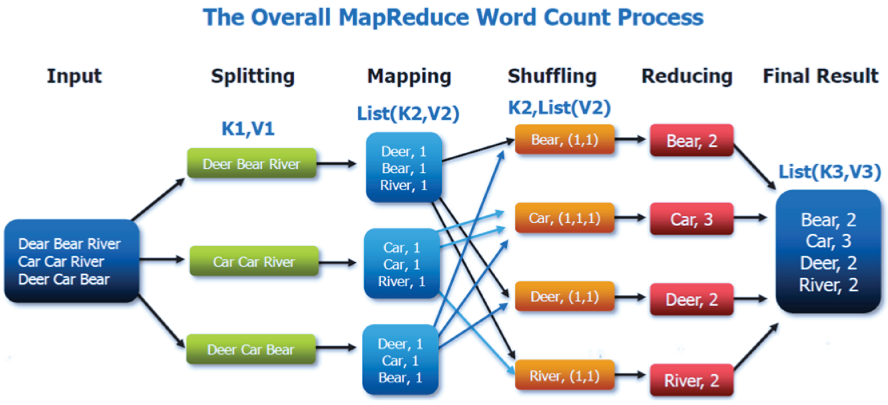


FIGURE 2.3: MapReduce word count example

Figure 2.3 presents a MapReduce example of counting words[2] . The input is a set of documents, which is probably stored on a distributed file system. There are three map tasks and four reduce tasks. Once a worker is assigned a map task, it reads one piece of the input, parses the content and outputs intermediate key-value pairs. In this example, an intermediate key is a word, and its value is its number of occurrences. After the intermediate pairs are are generated and buffered. A sorting (shuffling) phase takes place so that the numbers of occurrences for the same word are grouped together as a list. Then a worker that performs the reduce task iterates over the sorted intermediate

[2] `http://www.lebigdata.fr/mapreduce-tout-savoir`

key-value pairs and sums up the list of occurrences for each word. The final output is a set of word-occurrences pairs.

MapReduce is an innovative programming model that is capable to process large datasets on a cluster of commodity computers. However, it suffers from considerable I/O overhead, as the intermediate key-values pairs are unavoidably written to and read from the local disk. When it performs an iterative algorithm, the processing can be greatly slowed down, as the output of the preceding iteration is stored on local disk and then read in the following iteration. Due to this reason, MapReduce is not an ideal platform for iterative algorithms.

2.1.2.2 Apache Spark

Apache Spark is an in-memory distributed computing architecture for iterative and interactive applications. The core concept of Spark are RDDs [21], each of which is an immutable collection partitioned across a cluster of nodes. An RDD can be cached in memory. This mechanism allows each node to cache its respective slices for local computation and reuses them in other operations. The advantage of caching RDDs is that it avoids much I/O overhead. There are two operations supported by RDD abstraction: transformations and actions. An input dataset is firstly converted into a set of RDDs. By applying transformation functions, the input RDDs are transformed and this process generates a lineage of RDDs. Transformations are lazy functions, they do not execute until an action function is called. An action function executes all the transformation functions that construct the RDDs' lineage and returns final results, which are displayed or stored on local disk. Table 2.1 lists a few transformations and actions functions.

Transformations	Actions
map(func)	reduce(func)
flatMap(func)	collect()
filter(func)	count()
groupByKey()	first()
reduceByKey()	take(n)
sortByKey()	saveAsTextFile(path)
mapValues(func)	counteByKey()
union()	foreach(func)
distinct()	collectAsMap()
...	...

Table 2.1: A few RDDs' transformations and actions functions

The properties of RDDs make Spark an ideal programming model for bulk iterative algorithms [21]. These properties are:

- RDDs provide an interface on coarse-grained transformations that apply the same operation to many data items in parallel. Transformations are logged to built a dataset's lineage rather than operated on the actual data.
- RDDs are not materialized at all times. Instead, an RDD just has enough information about how it was derived from its lineage to compute its partitions from data in stable storage. Until an action is called, RDD will not materialize the storage data.
- RDDs can be cached in memory for further use. By default, all RDDs are kept in memory until memory is full, the least frequently-used RDDs are flushed into storage.

Along the growth of RDDs' lineage, the types of RDDs vary depending on the applied transformation functions. Figure 2.4 illustrates an example of a RDDs' lineage and Table 2.2 lists a few commonly-used RDD types with the corresponding transformation functions.

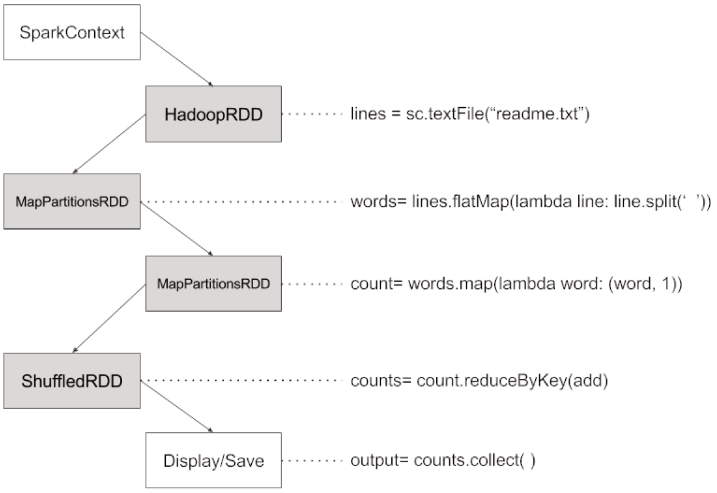


FIGURE 2.4: Growth of RDDs' lineage when counting words in PySpark commands

Once SparkContext initializes a Spark job, a data object can be encapsulated as a set of RDDs, on which different transformation functions operate. The lineage is only materialized upon an action function. If data is read from the Hadoop File System (HDFS), it becomes a set of HadoopRDDs, functions like map(), flatMap and filter() will transform it into MapPartitionRDDs. If user requires to change the number of parallel pieces of the RDDs, s/he can use the repartition() or the coalesce() function. This results in CoalescedRDDs. When working on key-value pairs, functions such as groupByKey(), join() and zip() function output PairRDDs.

Type	Description
HadoopRDD	result of reading from HDFS via function textFile()
MapPartitionRDD	result of calling map(), flatMap(), filter(), etc
CoalescedRDD	result of repartition() or coalesce() transformations
PairRDD	result of groupByKey(), join(), zip(), etc

TABLE 2.2: Types of commonly-used RDDs

In Spark, a user program is an application. Each application is composed of a number of jobs, and each job has a number of tasks. In terms of executing an application, Spark provides two modes: the local mode and the cluster mode. When Spark launches an application in local mode, all the tasks are actually run on one machine but in a multi-threading way. This mode is useful in developing, testing and debugging programs using small datasets. Cluster mode is deployed when a cluster of nodes is configured. Depending on the types of resource manager, there three cluster modes: standalone mode, Spark on Apache Mesos, and Spark on Hadoop YARN. The standalone mode is Spark's built-in cluster environment. It is available in the default distribution of Spark and it is the easiest way to run a Spark application on a top of a cluster. Apache Mesos and Hadoop Yarn are two different resource managing softwares. Mesos is built to be a global resource manager for an entire data center. It determines what resources are available when a job request comes into the Mesos master. Once the job request is accepted, it places the job on the workers. Hadoop YARN (short for Yet Another Resource Negotiator) is the second-generation MapReduce. It is a central resource manager for a Hadoop cluster. It monolithic schedules jobs after evaluating all available resources.

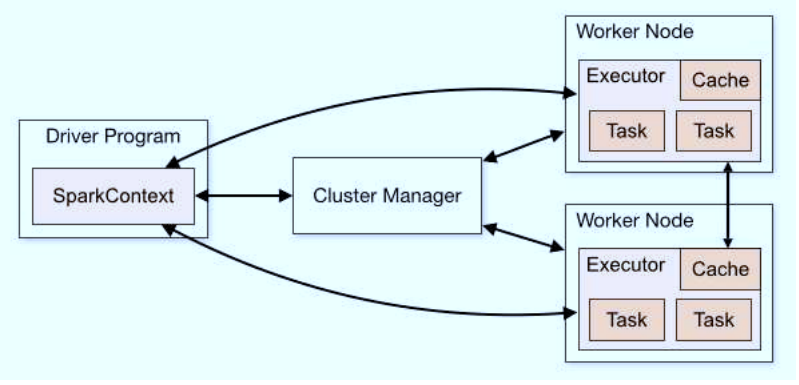


FIGURE 2.5: Spark cluster architecture

Figure 2.5 illustrates the structure of a cluster of nodes when a cluster mode is deployed[3]. It is composed of a driver program and several worker nodes, which are linked by a cluster manager (this can be the Spark built-in cluster manager or Mesos or YARN). In a cluster mode, an application runs as an independent set of processes on a cluster,

[3] `https://spark.apache.org/docs/latest/cluster-overview.html`

these processes are coordinated by the SparkContext, which is initialized in the driver program. Executors run the processes and store data on worker nodes. With the cluster manager allocating resources, Spark acquires executors from cluster nodes and sends user program to the executors, on which the user program is split into tasks, each task is a unit process[4].

Spark provides two sets of Machine Learning libraries (MLlib), the RDD-based[5] and the DataFrame-based[6]. A Spark DataFrame[7] is a distributed collection of data that is organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as existing RDDs, structured data files and tables in Hive[8]. Starting from Spark 2.0, the RDD-based MLlib entered maintenance mode and new features are only added to DataFrame-based MLlib.

From the production point of view, it is more efficient to use DataFrame-based MLlib in a business project, as the built-in ML functions allow users to quickly implement standard pipelined-programs without digging into implementation details like using the RDD-based libraries. However, these functions leave users little control over the data, and user programs are limited to a relatively small set of types and operations. Besides, for Python programmers, DataFrame-based ML functions are heavily wrapped in Java, this makes it hard to change parameters in a function or change it in order to suit a specific need. For functions that are not yet implemented in the DataFrame-based MLlib, it is challenging to implement user-defined programs. The RDD-based MLlib, on the other hand, contains more fine-grained functions and operations. Though more details have to be considered along implementation, it provides users more control on data and it allows users to implement their own algorithms using RDDs. Due to this reason, we choose to implement the proposed methods using Spark RDDs instead of DataFrames.

2.1.2.3 MapReduce v.s. Spark

Figure 2.6 and Figure 2.7 illustrate the major difference between Hadoop MapReduce and Apache Spark when they perform an iterative algorithm[9]. In the former platform, output that is generated in a preceding iteration is stored on HDFS, and the following iteration reads from HDFS to continue the computation. It is obvious that much I/O overhead is produced, which can largely slow down the computation. Differently, on the

[4] `https://spark.apache.org/docs/latest/cluster-overview.html`
[5] `https://spark.apache.org/docs/latest/mllib-guide.html`
[6] `https://spark.apache.org/docs/latest/ml-guide.html`
[7] `https://spark.apache.org/docs/latest/sql-programming-guide.html`
[8] `https://hive.apache.org/`
[9] `https://www.tutorialspoint.com/spark_sql/spark_sql_quick_guide.htm`

Spark platform, the intermediate output is stored in the distributed memory, allowing immediate data access for the following iterations. In this way, the computing time can be substantially shortened. This is one principle reason that makes Spark more advantageous than MapReduce in running an iterative algorithm.

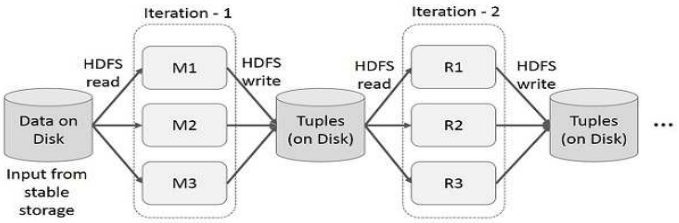


FIGURE 2.6: When MapReduce runs an iterative job

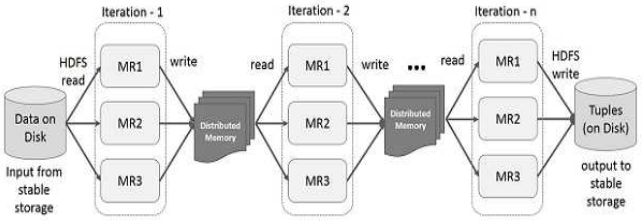


FIGURE 2.7: When Spark runs an iterative job

2.1.2.4 Distributed Storage Systems

Distributed Storage Systems are constructed by a number of network-connected inexpensive storage devices to store a massive amount of data with reliability and ubiquitous availability [25]. Examples of prevailing applications are Distributed Hash Table (DHT), Google File System (GFS) and Hadoop File System (HDFS).

HDFS has a master-worker architecture. An HDFS cluster is composed of a single NameNode and a number of DataNodes. The NameNode is the master server that managers the file system namespace and regulates access to files like opening, closing, and renaming files and directories. A DataNode is a worker node that manages its own storage. A file that is stored on HDFS is internally split into one or more blocks and these blocks are stored in a set of DataNodes. It is the NameNode that determines the mapping of blocs to DataNodes. The DataNodes, on the other hand, are responsible for reading and writing requests and for performing block creation, deletion and replication upon instruction from the NameNode [10].

[10] `https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`

To a user, access to files stored on HDFS is like accessing files stored on a single machine. Commands[11] that operate on HDFS files are similar to Linux commands.

2.2 Agglomerative Hierarchical Clustering

2.2.1 Overview

Clustering is a major unsupervised machine learning method. As shown in Figure 2.8 [2], clustering can be categorized into hard and fuzzy clustering. In hard clustering, each data instance belongs to one cluster; in fuzzy clustering, on the other hand, a data instance can be possessed by several clusters. Hard clustering further contains partitional and hierarchical clustering. Partitional clustering is also called flat clustering, it outputs individual clusters that do not connect to each other. K-means is an example of flat clustering. Hierarchical clustering outputs a binary tree-like structure, called dendrogram (Figure 2.9). A dendrogram explicitly exhibits the connections among clusters, i.e., how data instances are grouped iteratively. This presentation is advantageous in revealing interconnections of data instances and sub-clusters. A classic example of dendrogram is the revolutionary tree that exhibits the evolutionary relationship among various biological species, based on similarities and differences in their physical or generic characteristics.

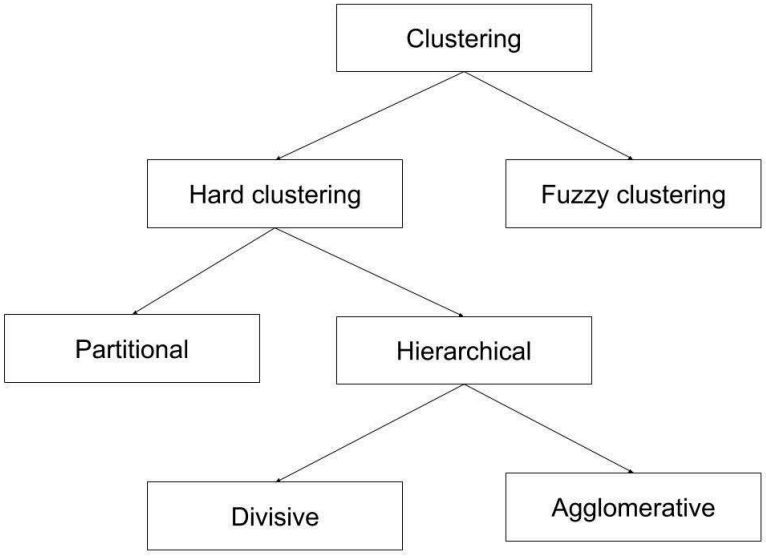


FIGURE 2.8: Overview of clustering methods

There are two types of hierarchical clustering, the divisive (DHC) and the agglomerative (AHC), both need to go through a number of iterations, but in different manners. Given

[11] `https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html`

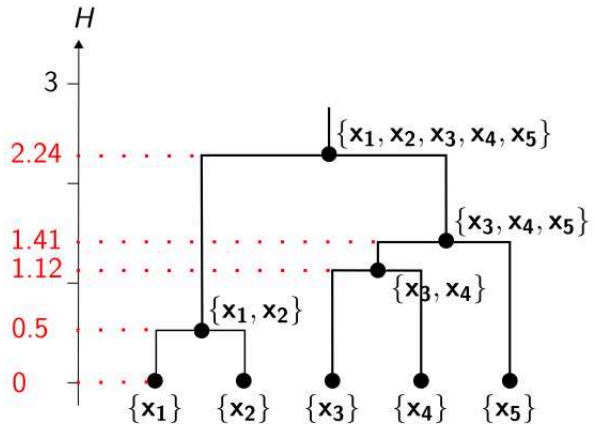


FIGURE 2.9: Example of dendrogram

a dataset $\mathcal{D}$ of n instances, DHC begins with splitting $\mathcal{D}$ into two clusters, and iteratively splits the generated clusters until each data instance becomes a cluster itself. It is a top-down process. AHC functions in the opposite way. It starts with n individual clusters, and merges a pair of clusters that have the minimal distance in each iteration. Eventually, it outputs a cluster of n instances. This is a bottom-up process. In terms of complexity, though AHC can reach $\mathcal{O}(n^3)$ in the worst case, it is usually favored over DHC, which is more computationally demanding. Because in each iteration, DHC has to find the best split among $2^{n-1} - 1$ possibilities. When n is large, finding an optimal split can be NP-hard. It is the reason why we chose AHC over DHC in this thesis.

Many AHC methods were proposed in the past decades. In principle, these methods can be categorized into:

- conventional AHC methods,
- nearest neighbor chain algorithms, and
- other approaches.

Apart from those methods that try to reduce the time complexity of AHC by algorithmic optimization, other methods try to improve computing efficiency by utilizing the technologies of distributed and parallel computing. According to the specific techniques, these methods can be classified into parallel methods and distributed methods.

2.2.2 Conventional Methods and the Lance-Williams Formula

2.2.2.1 Conventional Methods

Given a dataset of n instances, the general procedure of conventional AHC is shown in Algorithm 1. The output of this procedure is a dendrogram that grows upwards.

Algorithm 1 General procedure of AHC

Data: Pairwise dissimilarity matrix D of input data

Initialize a dendrogram of n leaves with null height values

while *number of iterations* $< n$ **do**

1. $(C_i, C_j) = \arg\min_{(C_x, C_y)} D(C_x, C_y)$, i.e. search for the minimal distance in D and the pair of clusters (C_i, C_j),
2. merge C_i and C_j into C_{ij} and add a corresponding parent node in the dendrogram with height value $D(C_i, C_j)$,
3. compute distance between C_{ij} and another cluster C_k, and update D accordingly.

end

Result: A dendrogram of $2n - 1$ nodes

There are different distance measures to compute $D(C_i, C_j)$, some of the commonly-used measures for numeric and for binary data are listed in Table 2.3, where $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_m)$ denote two vectors of m features. Note that the Minkowski distance becomes Manhattan distance, Euclidean distance and maximum distance, when its parameter p takes values of 1, 2 and ∞, respectively.

Data type	Distance	Formula
Binary	Jaccard distance	$D_{jac}(X, Y) = \lvert X \triangle Y \rvert / \lvert X \cup Y \rvert$
	Dice distance	$D_{dice} = \lvert X \triangle Y \rvert / (\lvert X \rvert + \lvert Y \rvert)$
Numeric	Minkowski distance	$D_{min}(X, Y) = \left(\sum_{j=1}^{m} \lvert x_j - y_j \rvert^p \right)^{1/p}$
	Mahalanobis distance	$D_{mah}(X, Y) = \lVert X - Y \rVert_A = \sqrt{(X - Y)A(X - Y)^T}$
	Average distance	$D_{ave}(X, Y) = \left(\frac{1}{m} \sum_{j=1}^{m} (x_j - y_j)^2 \right)^{1/2}$

TABLE 2.3: Commonly-used distance measures for numeric and binary data [4]

In the vector space model, when documents are vectorized using the TF-IDF weighting system, values are all numeric and non-negative. Under this model, documents are actually projected into the feature space, where the projection is sphere-like. Therefore, the Euclidean distance is usually chosen to determine the dissimilarity of two document vectors. And it is our choice to compute $D(C_i, C_j)$ in the general procedure of AHC.

In AHC, an essential step is to compute $D(C_{ij}, C_k)$ after C_i and C_j are merged. Shown in Figure 2.10, there are two types of methods, the graphic and the geometric. The former type uses cluster graphic representations to compute $D(C_{ij}, C_k)$, the single link method, the complete link method, the group average method and the weighted group average method belong to this type. The group average method is also named as average link or UPGMA (unweighted pair group method with arithmetic mean), and its weighted variation is the weighted group average method, which is also referred as the McQuitty method or WPGMA (weighted pair group method with arithmetic mean). While for the geometric methods, they use cluster centroids to determine the distance between C_{ij} and C_k. The centroid method, the median method and the Ward method are of this type. Table 2.4 lists the distance updating formulas of these methods, where $\overrightarrow{c}_{ij}$ denotes the centroid of cluster C_{ij}, i.e. $\overrightarrow{c}_{ij} = \frac{|i|\overrightarrow{c}_i + |j|\overrightarrow{c}_j}{|i|+|j|}$, with $|i|$ representing the number of members in cluster C_i, and $\overrightarrow{\omega}_{ij}$ denotes the median of cluster C_{ij} with $\overrightarrow{\omega}_{ij} = \frac{1}{2}(\overrightarrow{\omega}_i + \overrightarrow{\omega}_j)$.

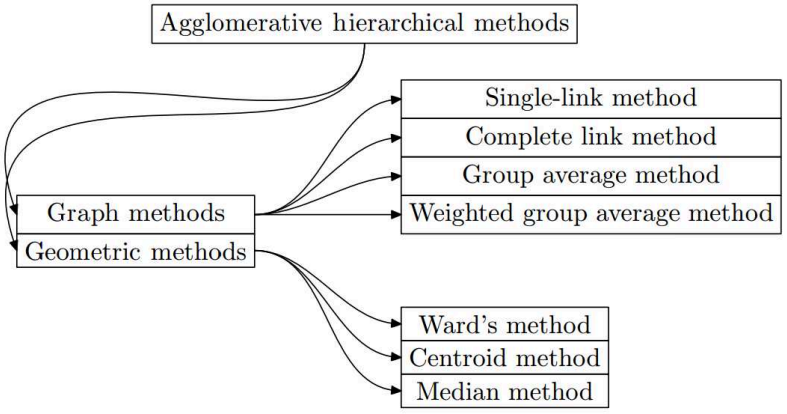


FIGURE 2.10: Commonly used conventional AHC methods [2]

Type	Method	Distance updating formula	$D(C_{ij}, C_k)$
Graphic	single	$min(d(i,k), d(j,k))$	$\min_{x \in C_{ij}, y \in C_k} D(x,y)$
	complete	$max(d(i,k), d(j,k))$	$\max_{x \in C_{ij}, y \in C_k} D(x,y)$
	average	$(n_i d(i,k) + n_j d(j,k))/(n_i + n_j)$	$\frac{1}{\lvert C_i \rvert \lvert C_j \rvert} \sum_{x \in C_{ij}, y \in C_k} D(x,y)$
	weighted	$(d(i,k) + d(j,k))/2$	
Geometric	centroid	$\sqrt{\frac{n_i d(i,k) + n_j d(j,k)}{n_i + n_j} - \frac{n_i n_j d(i,j)}{(n_i + n_j)^2}}$	$\lvert\lvert \overrightarrow{c}_{ij} - \overrightarrow{c}_k \rvert\rvert^2$
	median	$\sqrt{\frac{d(i,k)}{2} + \frac{d(j,k)}{2} - \frac{d(i,j)}{4}}$	$\lvert\lvert \overrightarrow{\omega}_{ij} - \overrightarrow{\omega}_k \rvert\rvert^2$
	Ward	$\sqrt{\frac{(n_i + n_k)d(i,k) + (n_j + n_k)d(j,k) - n_k d(i,j)}{n_i + n_j + n_k}}$	$\sqrt{\frac{2\lvert C_{ij} \rvert \lvert C_k \rvert}{\lvert C_{ij} \rvert + \lvert C_k \rvert}} \cdot \lvert\lvert \overrightarrow{c}_{ij} - \overrightarrow{c}_k \rvert\rvert^2$

TABLE 2.4: Graphic and geometric methods for computing $D(C_{ij}, C_k)$ [5]

2.2.2.2 The Lance-Williams Formula

Lance and Williams [26] formulate the above-mentioned methods in a unified way and name this formulation as the Lance-Williams formula. Equation 2.1 and Table 2.5 present this formula and list the methods and their parameter values.

$$\begin{aligned} D(C_{ij}, C_k) =& \alpha_i D(C_i, C_k) + \alpha_j D(C_j, C_k) + \beta D(C_i, C_j) \\ &+ \gamma |D(C_i, C_k) - D(C_j, C_k)| \end{aligned} \tag{2.1}$$

Methods	α_i	α_j	β	γ
single	$1/2$	$1/2$	0	-1/2
complete	$1/2$	$1/2$	0	$1/2$
average	$\frac{\lvert C_i\rvert}{\lvert C_i\rvert+\lvert C_j\rvert}$	$\frac{\lvert C_j\rvert}{\lvert C_i\rvert+\lvert C_j\rvert}$	0	0
McQuitty	$1/2$	$1/2$	0	0
centroid	$\frac{\lvert C_i\rvert}{\lvert C_i\rvert+\lvert C_j\rvert}$	$\frac{\lvert C_j\rvert}{\lvert C_i\rvert+\lvert C_j\rvert}$	$-\frac{\lvert C_i\rvert\lvert C_j\rvert}{(\lvert C_i\rvert+\lvert C_j\rvert)^2}$	0
median	$1/2$	$1/2$	-1/4	0
Ward	$\frac{\lvert C_i\rvert+\lvert C_k\rvert}{\lvert C_i\rvert+\lvert C_j\rvert+\lvert C_k\rvert}$	$\frac{\lvert C_j\rvert+\lvert C_k\rvert}{\lvert C_i\rvert+\lvert C_j\rvert+\lvert C_k\rvert}$	$-\frac{\lvert C_k\rvert}{\lvert C_i\rvert+\lvert C_j\rvert+\lvert C_k\rvert}$	0

TABLE 2.5: Lance-Williams formula: methods and parameter values

This formula provides much convenience in computing $D(C_{ij}, C_k)$ using the seven conventional AHC methods, which can be applied by simply fitting corresponding parameter values into the formula. However, it does not change the complexity of storage and computation. For these methods, the computation requires $O(n^2)$ storage for n initial objects, and $O(n^3)$ time complexity [27].

2.2.3 Nearest Neighbor Chain Approaches

Over decades, many AHC algorithms have been proposed for better efficiency. These include Sibson's SLINK algorithm [28], Rohlf's MST (Minimum Spanning Tree) algorithm [29] for single link, Defays' method for complete link method [30], and the reciprocal nearest neighbor methods of de Rham [31] and Juan [32]. Murtagh surveys these algorithms in [27, 33] and discusses their time and storage complexity. According to his survey, methods that use nearest neighbor chain to find reciprocal nearest neighbors have $O(n^2)$ time and storage complexity. An exception is the storage complexity of the Ward method, which is $O(n)$. Though with lower time complexity, the nearest neighbor chain approaches are not applicable for the centroid and the median methods.

Applying nearest neighbor chain (NN-chain) to perform AHC consists of two steps: (1) construct the NN-chain for each data instance and (2) look for the pairs of points that are reciprocal or mutual nearest neighbors (RNNs). Once the RNNs are found, they are

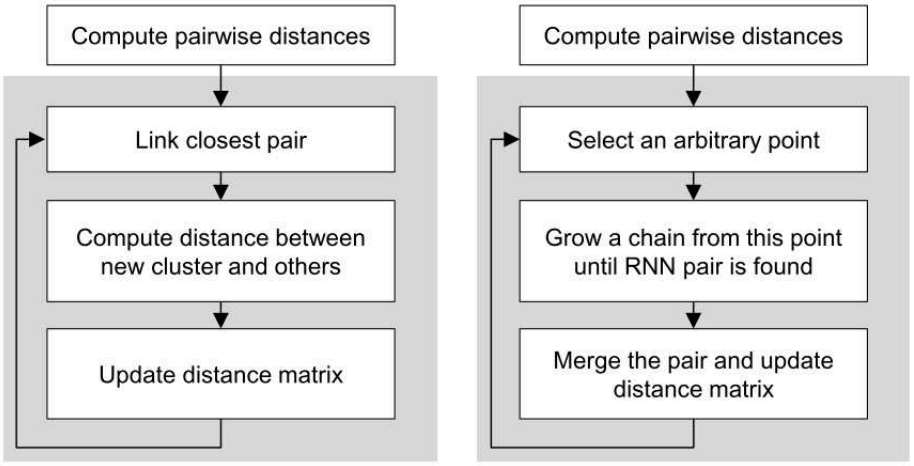


FIGURE 2.11: Comparison of conventional AHC methods (left) and the NN-chain algorithm (right) [3]

agglomerated as one cluster. The computing procedure of such a NN-chain algorithm is shown in Algorithm 2.

Algorithm 2 Computing procedure of the NN-chain algorithm [27]

Data: Pairwise dissimilarity matrix D of input data
Initialize a dendrogram of n leaves with null height values
while *number of iterations* $< n$ **do**
1. select an artitrary data point,
2. grow the NN-chain from this point until a pair of reciprocal nearest neighbors is obtained,
3. agglomerate these points by replacing them with a cluster point, add one node in the dendrogram, and update the dissimilarity matrix.

end
Result: A dendrogram of $2n - 1$ nodes

Differing from the conventional AHC methods that globally select and merge the pair of clusters that have the minimal distance, the NN-chain algorithm looks for the pair of reciprocal nearest neighbors in a chain, which is a local structure. A comparison of the two approaches is illustrate in Figure 2.11.

The NN-chain algorithm is claimed to produce the same results for five conventional AHC methods, they are single link, complete link, average link, McQuitty and Ward methods. These clustering methods do not generate height inversions in their dendrograms, i.e., the height of a child node is always lower than the height of its parent nodes. This can be formulated as:

$$D(C_i, C_j) < D(C_i, C_k) \text{ or } D(C_i, C_j) < D(C_j, C_k) \implies D(C_i, C_j) < D(C_{ij}, C_k) \quad (2.2)$$

It is a form of Bruynooghe's reducibility property [34]. A clustering method, whose dissimilarity matrix satisfies this property, produces a dendrogram shown as the left sub-figure in Figure 2.12. The single link, complete link, average link, the McQuitty and the Ward methods all output dendrograms of this type. The centroid and median methods, on the other hand, output dendrograms with height inversions, shown as the right sub-figure. Therefore, results of NN-chain output by these two methods cannot be guaranteed to have the same hierarchy as the conventional approaches.

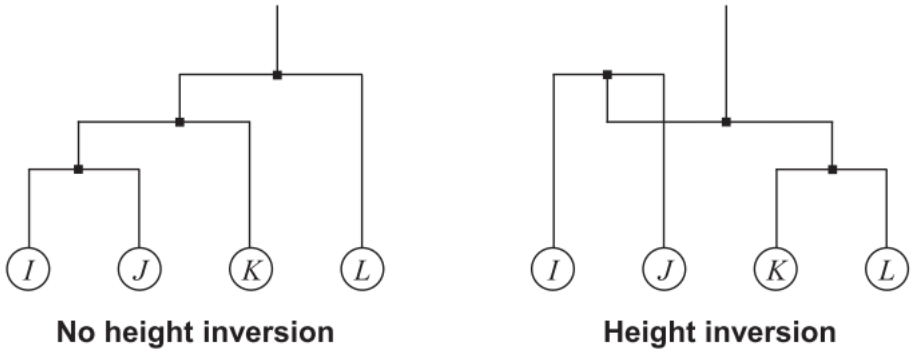


FIGURE 2.12: Dendrograms without and with height inversion

Müllner [5] proves the correctness of the NN-chain approach. Furthermore, he proposes a generic algorithm that overcomes the inversion problem for the median and centroid methods, based on Anderberg's idea of maintaining a list of nearest neighbors for each data point [35]. The major improvement of this algorithm is the transformation of the list of nearest neighbors into a cached priority queue. With such a structure, the time used for repeated minimum searches is reduced. But this modification does not really improve the complexity for the median and centroid methods. It is claimed to have a time complexity of $O(n^3)$ for the median and centroid methods, and $O(n^2)$ for the other methods unified by the Lance-Williams formula. As to the storage complexity, it is $O(n^2)$ for all. A shortcoming of this generic approach is that, for the geometric clustering methods, they are limited to the Euclidean distance. Plus, in this approach does not explain how the inversion problem is solved for the centroid and median methods.

Fast-RNN is another work that improves the efficiency of the NN-chain approach [36]. This work adopts the RNN clustering algorithm [37], which has $O(n^2m)$ time and $O(n)$ storage complexity. n and m denote the number of data instances and the number of features, respectively. The main innovation of fast-RNN algorithm is that, its NN-chain is constructed by a dynamic space partitioning strategy using slices. Given a predefined distance parameter ϵ, an iterative slicing procedure is applied to search for a list of nearest neighbor candidates for each data point. The candidates are interesting points that are located within a slice of width 2ϵ centered at a data point in a multi-dimensional space.

Attaching a candidate list of nearest neighbors to data point is a similar data structure used in [5]. However, the way of updating proximity between a newly merged cluster and a remaining cluster differs in fast-RNN. It employs a similarity measure that uses cluster centroids. This requires tracking the mean values and variances of clusters along the clustering procedure. Though fast-RNN returns identical results as the conventional RNN algorithm with better efficiency, it has some obvious constraints. One constraint is that it needs an extra step to find a proper value for ϵ, which does not seem to be a trivial task. Besides, this method is only shown to be applicable for the average link clustering method. Limited by its proximity measure, this method is not suitable for graph methods.

2.2.4 Approaches for Large Datasets

Aside from algorithms that focus to decrease the complexity of AHC, there are other approaches that try to make AHC applicable for large datasets. A well-known algorithm is BIRCH [6], which is capable to handle large datasets and is robust against outliers. The key feature of BIRCH is its use of a clustering feature (CF) tree, which is designed to capture important clustering information in the original data while requiring less storage. After the CF tree is constructed, an AHC algorithm is applied to the set of summaries to perform global clustering. BIRCH achieves a computational complexity of $O(n)$. Nevertheless, as BIRCH is independent from the Lance-Williams formula, it does not provide any method that is included in the formula.

Another famous algorithm is CURE [38], which applies a set of well-scattered points to represent a cluster. In this way, rich cluster shapes (other than hyperspheres) can be discovered. In addition, the chaining effect of minimum spanning tree, as well as the tendency to favor clusters with similar centroids can be avoided. CURE reduces computational complexity with a random sampling and partitioning strategy. Thus, its complexity is dependent on the size of the sampled dataset. Concretely, its time complexity is $O(n_{sample}^2 \log n_{sample})$ and its space complexity is $O(n_{sample})$. The main drawback of CURE is that its results are indeterministic due to its sampling procedures.

2.2.5 On-line AHC Algorithms

Apart from clustering on static datasets, some algorithms are capable to perform AHC on on-line datasets. These algorithms include MCUPGMA [39] for average link, ESPRIT hcluster [40] for single link and complete link, and SparseHC [41] for single link, complete link and average link.

Claimed to be a memory-efficient on-line algorithm, SparseHC scans a stored distance matrix chunk-by-chunk, compresses the information in the currently loaded chunk, then applies adjacency map (an efficient graph representation) to store unmerged cluster connections. It permits constant access to these connections for clustering with reduced computation time. This strategy empirically allows SparseHC to achieve a linear storage complexity. According to experimental results, SparseHC does decrease the memory growth. However, in terms of time complexity, only the efficiency of single link is shown improved.

2.2.6 Distributed and Parallel Approaches for AHC

There are various approaches that adopt distributed or parallel computing to speed up AHC. In literature, we can find parallel approaches that employ GPUs [42], MPI [43], multi-threading [3] and multi-processing [44]. For distributed approaches, there are [45, 46] and [47] that use MapReduce and Spark, respectively.

2.2.6.1 Parallel Approaches

Shalom et al. [42] implement parallel single link and complete link clustering methods on GPU using CUDA (compute unified device architecture). Invited by NVIDIA[12], CUDA is a parallel computing platform and application programming interface (API) that allow software developers to use GPU for general purpose processing. Du and Lin [43] parallelize the single link method based on MPI for multiple-instruction and multiple-data environments. Jeon and Yoon [3] propose a parallelizing scheme for multi-threaded shared-memory machines to alleviate the cost of performing AHC, based on the concept of nearest neighbor chains. The proposed method allocates available threads into two groups, one for managing nearest neighbor chains, and the other for updating distance information. Hendrix et al. [44] present SHRINK, a scalable algorithm that implements the single link method in OpenMP (open multi-processing). OpenMP is an API that supports multi-platform shared memory multiprocessing programming. It uses a portable, scalable model that provides programmers a flexible interface to develop parallel applications on different platforms, which vary from standard desktop computers to supercomputers. The main parallelizing strategy of SHRINK is to divide the input dataset into overlapping subsets. Dendrogram is generated for each subset using the SLINK algorithm [28]. A full dendrogram is eventually constructed by combining small individual dendrograms.

[12] `http://www.nvidia.com/content/global/global.php`

2.2.6.2 Distributed Approaches

Wang and Dutta [45] present PARABLE, a hierarchical clustering algorithm for the MapReduce framework. It proceeds in two main steps: local hierarchical clustering on nodes using mappers and reducers, and integration of results by a dendrogram alignment technique. As this approach randomly splits data into smaller partitions in the first step, it only outputs approximate results. Jin et al. [46] formulate the single link method as an MST (minimum spanning tree) construction problem on a complete graph. This method firstly decomposes the complete graph into a set of non-overlapped subgraphs, then computes the intermediate sub-MSTs for each subgraph, and merges all sub-MSTs to output final result. Additionally, this approach can treat incremental data insertion for a separate data subset and integrate it with the existing result. The MapReduce implementation of this method achieves a significant speed up. However, evaluation of clustering quality is not given in this work. Later, Jin et al. [47] provide an extended work of [46] by offering another distributed implementation using the Spark platform. Compared with their former work, the authors argue that, the new implementation performs significantly better in terms of performance and scalability. Still, evaluation on clustering quality is not discussed.

2.3 Co-clustering

2.3.1 Overview

Co-clustering is a subspace clustering method, it performs clustering in both data and feature spaces by simultaneously partitioning data instances and their attributes into subgroups, which are called co-clusters. Each co-cluster contains a subset of data instances and a subset of attributes. For example, for a gene expression dataset, a co-cluster is composed of a subset of closely grouped genes and a subset of conditions that are highly associated to the subset of genes. For a collection of documents, co-clustering returns a subset of documents mixed with a subset of “describing" terms. In research works on co-clustering, it is also named as block clustering, direct clustering, cross-clustering, simultaneous clustering, bi-clustering, two-way clustering, two-mode clustering or two-side clustering [48]. In this thesis, we use the name of co-clustering.

For over four decades, co-clustering has been applied in many different domains such as text mining, bio-informatics, Web mining, etc. In text mining, a collection of documents is usually represented as a document-term matrix. As this matrix is often sparse and of high dimensions, co-clustering offers a good solution to reduce dimensions and to summarize the data. Two widely cited works are [9] and [49] that respectively apply spectral

bipartite graph partitioning and information-theoretic approach to find co-clusters in a document-term matrix. In bio-informatics, input to a co-clustering method is often a data matrix whose rows are gene expression levels and columns are experimental conditions. A common objective is to find subsets of genes whose expression levels exhibit a coherent pattern under subsets of conditions [48]. Madeira and Oliveira [50], Tanay et al. [51] provide comprehensive surveys that address this topic. In Web mining, co-clustering is an effective method in revealing subsets of Web users and pages that are closely related [52]. In marketing, especially in e-commerce, collaborative filtering-based recommender systems are widely applied to predict a user's preferred products by using other users' preferences. However, limited by the computational cost, most collaborative filtering methods can only be deployed in static off-line settings, where the information of known preference does not change over time. George and Merugu [53] propose a scalable on-line collaborative filtering framework based on a weighted co-clustering algorithm [54] that performs simultaneous clustering on users and products.

Based on various underlying assumptions, co-clustering approaches can be categorized into: (I) latent block models, (II) graph partitioning approaches, and (III) matrix factorization methods. We illustrate several representative algorithms in each category. As one of the proposed methods, SHCoClust, is based on the graph partitioning approaches, so we present more mathematical details, such as properties, deduction and reasoning, for the optimization problems involved in the graph partitioning approaches.

2.3.2 Latent Block Models

With an assumption that the values of a data matrix $A = \{a_{ij}\}_{i=1,\ldots,n;j=1,\ldots,m}$ are observations, and they are generated by a probabilistic generative model with underlying hidden blocks. Each block is composed of a partition of data instances z and a partition of data attributes w. The mapping between z and w is determined by a latent variable. By embedding co-clustering in such a probabilistic framework, Latent block models (LBMs) with several variations are proposed [55, 56, 57, 58, 59, 60].

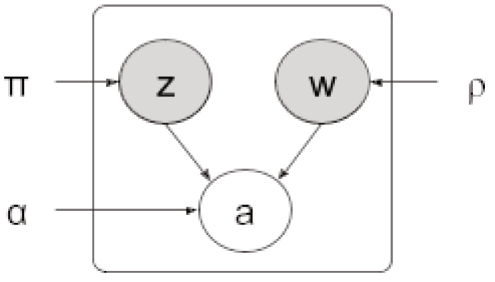


FIGURE 2.13: Latent block model as a graphical model

Figure 2.13 presents a graph model that illustrates the process of randomized data generation in LBMs. Determined by parameter $\boldsymbol{\alpha}$, a is an observed variable presented in the data matrix A. z and w are latent variables that are controlled by parameters $\boldsymbol{\pi}$ and $\boldsymbol{\rho}$. This process consists of:

- generating labelings $z = (z_1, \ldots, z_n)$ into g clusters according to the categorical distribution $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_g)$,
- generating labelings $w = (w_1, \ldots, w_m)$ into h clusters according to the categorical distribution $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_h)$, and
- generating a real value a_{ij} for $i = 1, \ldots, n$ and $j = 1, \ldots, m$ according to the distribution $f(.; \boldsymbol{\alpha}_{z_i w_j})$.

Let $\mathcal{Z}$ and $\mathcal{W}$ respectively denote the possible labels z for the set of data instances and w for the set of data attributes, the probability density function of a is defined as:

$$f(A; \boldsymbol{\theta}) = \sum_{(z,w \in \mathcal{Z} \times \mathcal{W})} p(z, w) f(A|z, w; \boldsymbol{\theta}) \tag{2.3}$$

with $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{\rho}, \boldsymbol{\alpha})$. Three methods are proposed to estimate parameters in Equation 2.3 using its log-likelihood, they are LBVEM (the variational EM approach), LBCEM (the classification EM approach) and LBSEM (the stochastic EM-Gibbs approach)[13].

LBVEM and LBCEM are two approximations to estimate $\boldsymbol{\theta}$ based on a distribution on $\mathcal{Z} \times \mathcal{W}$, denoted by $R = P(z, w|A, \boldsymbol{\theta})$. LBVEM imposes that the distribution of labels is assumed to be independent, i.e., $R(z, w) = R(z)R(w)$ with $R(z) = \prod_i q(z_i)$ and $R(w) = \prod_j q(w_j)$. By maximizing an approximation of the derived likelihood through a number of iterations, LBVEM obtains cluster labels for rows and for columns in A. LBCEM consists of inserting a classification step between E and M steps in estimating $\boldsymbol{\theta}$. Both LBVEM and LBCEM guarantee the convergence, and they are simple to implement and scalable [48].

LBSEM is a simple adaptation to LBVEM of the standard stochastic EM algorithm [61]. However, unlike the standard method that incorporates a stochastic step between the E and the M steps to simulate missing data based on their conditional distribution, LBSEM applies a Gibbs sampling scheme to simulate the couple (z, w). This is the characteristic that makes LBSEM and LBVEM different. Unlike LBVEM, LBSEM uses no approximation, and it is less sensitive to starting values.

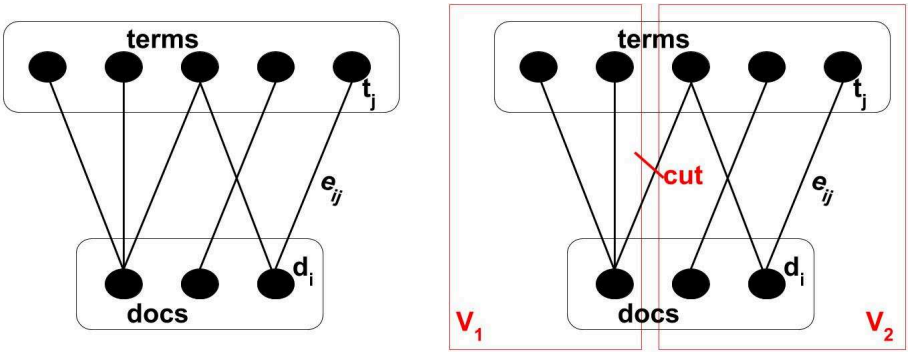
[13] EM is short for expectation-maximization.

Depending on the data type of matrix A, several variations of LBVEM and LBCEM are derived, with incorporating suitable data distributions. For example, when A is binary, there are Bernoulli LBVEM and Bernoulli LBCEM; when A is continuous, there are Gaussian LBVEM and Gaussian LBCEM algorithms; and when A is being considered as a contingency table, there are Poinsson LBVEM and Poisson LBCEM.

When A is being considered as a contingency table, Charrad et al. [62] proposes a method to determine the number of co-clusters, by alternatively applying K-means algorithm on rows and on columns to form a number of co-clusters that optimize their χ^2 values.

2.3.3 Graph Partitioning Approaches

Unlike LBMs, graph partitioning approaches model a data matrix as an undirected bipartite graph, in which two sets of vertices are connected by a number of edges. No edge occurs between two vertices of the same set, and every edge carries a sort of association between two connected vertices. This association is measure by a value of weight. Figure 2.14(a) illustrates an example, in which a bipartite graph models a small document-term matrix. Its vertices are composed of documents and terms, and each of its edges carries a TF-IDF value. With this model, finding co-clusters in a data matrix is equivalent to partitioning its bipartite graph into several sub-graphs. Figure 2.14(b) displays a solution that partitions the bipartite in Figure 2.14(a) into two sub-graphs, V_1 and V_2. Each sub-graph contains a subset of documents and a subset of terms.



(a) (b)

FIGURE 2.14: An illustration of (a) a bipartite graph and (b) its partitions. d_i denotes a document, t_j denotes a term, e_{ij} denotes an edge that links d_i and t_j. V_1 and V_2 denote two sub-graphs.

The objective of a clustering method is to divide input data instances into several groups such that instances in the same group are similar and those in different groups are dissimilar [63]. Likewise, when partitioning a bipartite graph, we aim to obtain sub-graphs such that vertices and edges in the same sub-graph are tightly associated, meanwhile vertices and edges in different sub-graphs are loosely connected. To achieve this, we

need to partition the graph by removing the weak edges, that carry the least association between two sub-graphs. In graph theory, the concept that partitions a graph into two disjoint sub-graphs is *cut*. For two disjoint sub-graphs, V_1 and V_2, their *cut* is defined as

$$cut(V_1, V_2) = \sum_{i \in V_1, j \in V_2} e_{ij} \tag{2.4}$$

and for K disjoint sub-graphs, $V_1, \ldots, V_K$, their cut is defined as

$$cut(V_1, \ldots, V_K) = \sum_{i=1}^{K} cut(V_i, \overline{V_i})$$

with $\overline{V_i}$ denoting the complement of V_i, that is $\overline{V_i} \cup V_i = V$.

Von Luxburg [63] argues that when $K > 2$, the solution of *mincut* often results in separating one individual from the rest of the graph. It outputs sub-graphs of very unbalanced sizes. In practice, sub-graphs are expected to be “reasonably large". There exist two common objective functions to encode the desired scenarios: the RatioCut [64] and the normalized cut *Ncut* [65], defined in Equation 2.5 and in Equation 2.6, respectively:

$$RatioCut(V_1, \ldots, V_K) = \sum_{i=1}^{K} \frac{cut(V_i, \overline{V_i})}{|V_i|} \tag{2.5}$$

$$Ncut(V_1, \ldots, V_k) = \sum_{i=1}^{K} \frac{cut(V_i, \overline{V_i})}{vol(V_i)} \tag{2.6}$$

where $|V_i|$ denotes the number of vertices in V_i, and $vol(V_i)$ denotes the sum of degrees of all vertices in V_i. The degree of a vertex is the sum of values of all its edges. The sets of sub-graph vertices satisfy $V_i \cap V_j = \emptyset$ and $V_1 \cup \cdots \cup V_K = V$. Equations 2.5 and 2.6 have small values if the sub-graphs V_i are not too small. When $cut(V_i, \overline{V_i}) = 1$, Equations 2.5 and 2.6 respectively reach the minimum if all $|V_i|$ or all $vol(V_i)$ coincide. Both scenarios result in balanced sub-graphs, measured by the number of vertices or edge weights. However, minimizing either function with balancing conditions is a NP-hard task [66]. Spectral clustering [63] provides a solution to solve relaxed versions of these problems. The co-clustering methods proposed in [9, 67] detail two approaches that use normalized spectral clustering by relaxing *Ncut*. A derived approach that applies an isoperimetric embedding is explained in [68]. As these approaches output the same number of row-wise and column-wise clusters, resulting co-clusters are along the diagonal of the input data matrix.

In order to have a better understanding on how spectral clustering can be used as a solution in graph partitioning to perform co-clustering, we present basic concepts of spectral clustering, its mathematical insight and some state-of-the-art approaches.

2.3.3.1 Graph Laplacians, Properties and Spectral Clustering

Let $G = (V, E)$ denote a bipartite graph constructed by a set of vertices $V = (v_i)_{i=1,\dots,n}$ and a set of edges E, and let $W = (w_{ij})_{i,j=1,\dots,n}$ denote the weighted adjacency matrix of G. $w_{ij} = 0$ means that vertices v_i and v_j are not connected, and $w_{ij} = w_{ji}$ as G is undirected. The degree of a vertex $v_i \in V$ is defined as $deg_i = \sum_{j=1}^{n} w_{ij}$, it is the sum that runs over all vertices adjacent to v_i. The degree matrix D is a diagonal matrix with degrees $deg_1, \dots, deg_n$ on the diagonal. As we start with $K = 2$, thus $V = V_1 \cup V_2$. Let $f = [f_1, \dots, f_n]^T$ denote the partition vector defined as

$$f_i = \begin{cases} +1, & i \in V_1, \\ -1, & i \in V_2. \end{cases} \tag{2.7}$$

Spectral clustering is favored in many applications over traditional clustering methods, as it can be solved efficiently by standard linear algebra software and has better clustering quality. The main tools for spectral clustering are graph Laplacian matrices. Von Luxburg [63] presents two types of graph Laplacian: unnormalized and normalized. The unnormalized graph Laplacian matrix is defined as $L = D - W$. Details on its properties can be found in [69, 70]. For normalized graph Laplacian, there are two forms of matrices. One is defined as $L_{sym} = D^{-1/2} L D^{-1/2}$ and the other is defined as $L_{rw} = D^{-1} L$. The former is a symmetric matrix, while the latter is closely related to random walk. Their properties are detailed in [71]. In [63], the author summarizes some general properties of unnormalized and normalized Laplacian matrices, and several properties of L_{sym} and L_{rw}. As these properties are essential in understanding spectral clustering methods and graph partitioning co-clustering approaches, we present them below:

- **General properties of graph Laplacian matrix.**

  1. For every $f \in \mathbb{R}^n$, there is

  $$f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (f_i - f_j)^2. \tag{2.8}$$

  2. L is symmetric and positive semi-definite.

3. The smallest eigenvalue of L is zero, and the corresponding eigenvector is the constant one vector, $e = [1, \ldots, 1]^T$.
4. L has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.

- **Main properties of the normalized graph Laplacian matrix.**

1. For every $f \in \mathbb{R}^n$, there is

$$f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \left(\frac{f_i}{\sqrt{deg_i}} - \frac{f_j}{\sqrt{deg_j}} \right)^2. \tag{2.9}$$

2. λ is an eigenvalue of L_{rw} with eigenvector v if and only if λ is an eigenvalue of L_{sym} with eigenvector $D^{1/2}v$.
3. λ is an eigenvalue of L_{rw} with eigenvector v if and only if λ and v solve the generalized eigenproblem $Lv = \lambda Dv$.
4. 0 is an eigenvalue of L_{rw} with the constant one vector e as eigenvector. 0 is an eigenvalue of L_{sym} with eigenvector $D^{1/2}e$.
5. L_{sym} and L_{rw} are positive semi-definite and have n non-negative real-valued eigenvalues $0 = \lambda_1 \leq \cdots \leq \lambda_n$.

- **From eigenvalue 0 of L to the connected components.**

The multiplicity[14] of the eigenvalue 0 of a graph Laplacian matrix is related to the number of connected components in the bipartite graph. Given G as an undirected graph with non-negative weights on all edges, the multiplicity K of the eigenvalue 0 of L equals the number of connected components $V_1, \ldots, V_K$ in the graph. For unnormalized L and for normalized L_{rw}, the eigenspace of eigenvalue 0 is spanned by the indicator vectors $e_{V_1}, \ldots, e_{V_K}$ of those components; for L_{sym}, the eigenspace of eigenvalue 0 is spanned by the vectors $D^{1/2}e_{V_1}, \ldots, D^{1/2}e_{V_K}$.

Von Luxburg [63] presents an intuitive explanation on the relation stated above. Assuming that all vertices in G are ordered according to the K connected components they belong to, the adjacency matrix W and the Laplacian matrix L thus have a block diagonal form such as

$$L = \begin{bmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_K \end{bmatrix}.$$

[14] The geometric multiplicity of an eigenvalue is the number of linearly independent eigenvectors associated with it.

Each block L_i is a proper graph Laplacian on its own, and it corresponds to the i_{th} connected component. The spectrum of L is given by the union of the spectra of L_i, and the corresponding eigenvectors of L are the eigenvectors of L_i. Every L_i has eigenvalue 0 with multiplicity 1, and the corresponding eigenvector is the constant one vector e on the i-th connected component. Therefore, L has as many eigenvalues 0 as there are connected components, and the corresponding eigenvectors are the indicator vectors of the connected components.

- **Unnormalized and normalized spectral clustering methods.**

  There exist many approaches that apply the properties of unnormalized and normalized Laplacian matrices to solve clustering problems. Assuming that the input dataset contains $x_1, \ldots, x_n$ instances, these approaches require a pairwise similarity matrix $S = (s_{ij})_{i,j=1,\ldots,n}$ with $s_{ij} = s(x_i, x_j)$ to construct a similarity graph. Three representative approaches are respectively represented by Algorithm 3, Algorithm 4 and Algorithm 5. The common point in these approaches is that they apply the K-means method on a transformed matrix constructed from the first K eigenvectors of a Laplacian matrix.

Algorithm 3 Spectral clustering using unnormalized L [63]

Require: Similarity matrix $S \in \mathbb{R}^{n \times n}$, K clusters to construct

1: Construct a similarity graph G from S, let W be the weighted adjacency matrix, and let D be the diagonal matrix.
2: Compute the unnormalized Laplacian $L = D - W$.
3: Compute the first K eigenvectors $v_1, \ldots, v_K$ of L.
4: Let $V \in \mathbb{R}^{n \times K}$ be the matrix containing the vectors $v_1, \ldots, v_K$ as columns.
5: For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^K$ be the vector corresponding to the i^{th} row of V.
6: Cluster the points $(y_i)_{i=1,\ldots,n}$ in $\mathbb{R}^K$ with the K-means algorithm into clusters $C_1, \ldots, C_K$.

Ensure: Clusters $V_1, \ldots, V_K$ with $V_i = \{j | y_j \in C_i\}$.

Algorithm 4 Spectral clustering using L_{rw} [65]

Require: Similarity matrix $S \in \mathbb{R}^{n \times n}$, K clusters to construct

1: Construct a similarity graph G from S, let W be the weighted adjacency matrix, and let D be the diagonal matrix.
2: Compute the first K eigenvectors $v_1, \ldots, v_K$ of the generalized eigenproblem $(D - W)v = \lambda Dv$.
3: Let $V \in \mathbb{R}^{n \times K}$ be the matrix containing the vectors $v_1, \ldots, v_K$ as columns.
4: For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^K$ be the vector corresponding to the i-th row of V.
5: Cluster the points $(y_i)_{i=1,\ldots,n}$ in $\mathbb{R}^K$ with the K-means algorithm into clusters $C_1, \ldots, C_K$.

Ensure: Clusters $V_1, \ldots, V_K$ with $V_i = \{j | y_j \in C_i\}$.

Algorithm 5 Spectral clustering using L_{sym} [72]

Require: Similarity matrix $S \in \mathbb{R}^{n \times n}$, K clusters to construct

1: Construct a similarity graph G from S, let W be the weighted adjacency matrix, and let D be the diagonal matrix.
2: Compute the normalized Laplacian $L_{sym} = D^{-1/2}(D - W)D^{-1/2}$.
3: Let $V \in \mathbb{R}^{n \times K}$ be the matrix containing the vectors $v_1, \ldots, v_K$ as columns.
4: Form the matrix $U \in \mathbb{R}^{n \times K}$ from V by normalizing the row sums to have norm 1, i.e., $u_{ij} = \frac{v_{ij}}{\sqrt{\sum_K v_{iK}^2}}$
5: For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^K$ be the vector corresponding to the i-th row of U.
6: Cluster the points $(y_i)_{i=1,\ldots,n}$ in $\mathbb{R}^K$ with the K-means algorithm into clusters $C_1, \ldots, C_K$.

Ensure: Clusters $V_1, \ldots, V_K$ with $V_i = \{j | y_j \in C_i\}$.

2.3.3.2 Mathematical Insight: Spectral Graph Partitioning by Optimizing *Ncut*

In order to understand how spectral clustering can solve graph partitioning by relaxing an optimization problem, we detail mathematical deduction and reasoning in this section. We explain several graph partitioning co-clustering approaches that apply *Ncut* and our reasoning focuses on *Ncut*. This reasoning explains the mathematical formulation in Section 2.3.3.3, which SHCoClust is established on. To make it simple, we consider the simplest case, when $K = 2$, i.e., it is only required to cut G into two parts, $V = V_1 \cup V_2$. Accordingly, the objective function of minimizing *Ncut* becomes

$$\min Ncut(V_1, V_2) = \frac{cut(V_1, V_2)}{vol(V_1)} + \frac{cut(V_2, V_1)}{vol(V_2)}. \tag{2.10}$$

In *Ncut*, an element in the partition vector f takes a form of

$$f_i = \begin{cases} \sqrt{\frac{vol(V_2)}{vol(V_1)}}, & if \quad i \in V_1, \\ -\sqrt{\frac{vol(V_1)}{vol(V_2)}}, & if \quad i \in V_2. \end{cases} \tag{2.11}$$

The optimization problem shown in Equation 2.10 is non convex. However, with the aid of Equation 2.8, it can be converted to a convex problem.

$$2f^T Lf = \sum_{i,j=1}^{n} x_{ij}(f_i - f_j)^2 \tag{2.12}$$

$$= \sum_{\substack{i\in V_1\\ j\in V_2}} x_{ij}\Big(\sqrt{\frac{vol(V_2)}{vol(V_1)}} + \sqrt{\frac{vol(V_1)}{vol(V_2)}}\Big)^2 + \sum_{\substack{i\in V_1\\ j\in V_2}} x_{ij}\Big(-\sqrt{\frac{vol(V_1)}{vol(V_2)}} - \sqrt{\frac{vol(V_2)}{vol(V_1)}}\Big)^2 \tag{2.13}$$

$$= \sum_{\substack{i\in V_1\\ j\in V_2}} x_{ij}\Big(\frac{vol(V_2)}{vol(V_1)} + \frac{vol(V_1)}{vol(V_2)} + 2\Big) + \sum_{\substack{i\in V_1\\ j\in V_2}} x_{ij}\Big(\frac{vol(V_1)}{vol(V_2)} + \frac{vol(V_2)}{vol(V_1)} + 2\Big)$$

$$= \Big(\sum_{\substack{i\in V_1\\ j\in V_2}} x_{ij} + \sum_{\substack{i\in V_1\\ j\in V_2}} x_{ij}\Big)\Big(\frac{vol(V_1)}{vol(V_2)} + \frac{vol(V_2)}{vol(V_1)} + 2\Big) \tag{2.14}$$

$$= 2cut(V_1, V_2)\Big(\frac{vol(V_1)}{vol(V_2)} + \frac{vol(V_2)}{vol(V_2)} + \frac{vol(V_2)}{vol(V_1)} + \frac{vol(V_1)}{vol(V_1)}\Big) \tag{2.15}$$

$$= 2cut(V_1, V_2)\Big(\frac{vol(V)}{vol(V_2)} + \frac{vol(V)}{vol(V_1)}\Big)$$

$$= 2vol(V)\Big(\frac{cut(V_1, V_2)}{vol(V_2)} + \frac{cut(V_1, V_2)}{vol(V_1)}\Big)$$

$$= 2vol(V)Ncut(V_1, V_2) \tag{2.16}$$

Note that in Equation 2.12, x_{ij} replaces w_{ij} because of the relation presented in Equation 2.22. With Equation 2.11, Equation 2.12 is extended to be Equation 2.13. In Equation 2.14, applying the definition of *cut* explained in Equation 2.4 and expanding the scalar 2 to be two more items results in Equation 2.15. Finally, given $K = 2$, $vol(V) = vol(V_1) + vol(V_2)$ and with the definition of *Ncut* shown in Equation 2.6, Equation 2.16 is obtained. As $vol(V)$ is a constant, the objective function presented in Equation 2.10 is in fact equivalent to

$$\begin{aligned} \min \quad & f^T Lf \\ s.t. \quad & Df \perp e, \\ & f^T Df = vol(V). \end{aligned} \tag{2.17}$$

Note that Equation 2.11 is another constraint for the new optimization problem presented in Equation 2.17. However, with this constraint, the entries of the solution vector f are only allowed to take two particular values, making the optimization problem NP-hard. A solution is to relax this constrain by allowing the entries in f to be reals, i.e., $f_i \in \mathbb{R}$. With this relaxation, the optimization problem can be solved with the aid of Laplacian

properties (Section 2.3.3.1). Let $f = D^{-1/2}g$, this substitution in Equation 2.17 results in

$$\begin{aligned} \min \quad & g^T D^{-1/2} L D^{-1/2} g \\ s.t. \quad & g \perp D^{1/2} e, \\ & \|g\|^2 = vol(V). \end{aligned} \tag{2.18}$$

where $D^{-1/2}LD^{-1/2} = L_{sym}$, $D^{1/2}e$ is the first eigenvector of L_{sym} and $vol(V)$ is a constant. Problem 2.18 is in the form of the standard Rayleigh-Ritz theorem (also named Rayleigh quotient). Solution g is given by the second eigenvector of L_{sym}, and f is in fact the second eigenvector of the generalized eigenproblem $Lv = \lambda Dv$.

2.3.3.3 Co-clustering Documents and Terms Using Spectral Graph Partitioning

Dhillon [9] proposes a co-clustering approach, in which a document collection is modeled as a bipartite graph and graph partitioning is applied to discover co-clusters. Let $A = (a_{ij})_{i=1,\ldots,n.j=1,\ldots,m}$ denote a document-term matrix that is extracted from a collection of documents. Each element of A is a TF-IDF score, i.e., $a_{ij} = t_{ij} \times \log\left(\frac{|n|}{|n_i|}\right)$, where t_{ij} denotes the frequency of term t_j in document d_i, $|n|$ denotes the total number of documents in the collection, and $|n_i|$ denotes the number of documents that contain term t_j. In this work, A is modeled as a bipartite graph G, whose vertices are documents and terms, and a TF-IDF score is considered as an association between a document and a term. An interesting point presented in this work is that the author presents the concept of duality of term clustering and document clustering, and he shows how this duality is related to graph partitioning. As this method outputs the same number of clusters on documents and on terms, let us assume that both document set I and term set J contain K clusters, with indexes $l = 1, \ldots, K$. A document cluster is denoted by z_l, with $z_1 \cup \cdots \cup z_K = I$ and $z_l \cap z_{l'} = \emptyset, l \neq l'$. A term cluster is denoted by w_l, with $w_1 \cup \cdots \cup w_K = J$ and $w_l \cap w_{l'} = \emptyset, l \neq l'$. By definition, "a given term t_j belongs to a term cluster w_l if its association with the document cluster z_l is greater than its association with any other document cluster" [9]. Therefore, there exists:

$$w_l = \left(t_j : \sum_{d_i \in z_l} a_{ij} \geq \sum_{d_i \in z_{l'}} a_{ij},\ l' = 1, \ldots, K \right). \tag{2.19}$$

Likewise, a document cluster can be defined as:

$$z_l = \left(d_i : \sum_{t_j \in w_l} a_{ij} \geq \sum_{t_j \in w_{l'}} a_{ij},\ l' = 1, \ldots, K \right). \tag{2.20}$$

Equations 2.19 and 2.20 illustrate that a given document clustering determines a term clustering, which in turn determines a better document clustering. This connection is a duality of term and document clustering. The “best" term and document clustering would correspond to a sub-graph such that the crossing edges between partitions have minimum weights. This is achieved when

$$cut(z_1 \cup w_1, \ldots, z_K \cup w_K) = \min_{V_1, \ldots, V_K} cut(V_1, \ldots, V_K). \tag{2.21}$$

As stated in Section 2.3.3, optimizing Equation 2.21 results in unbalanced clusters. Dhillon [9] avoids this problem by minimizing *Ncut* (Section 2.3.3.2). He starts with a bi-partitioning problem when $K = 2$, then extends the approach to multi-partitioning. However, different from the assumptions of spectral clustering (Section 2.3.3.1), this approach defines the adjacency matrix and the degree matrix respectively to be

$$W = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}, \quad D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \tag{2.22}$$

with $D_1(i,i) = \sum_j a_{ij}$ and $D_2(j,j) = \sum_i a_{ij}$, indicating the sum of edge-weights incident on document i and on term j, respectively.

This results in a Laplacian matrix

$$L = \begin{bmatrix} D_1 & -A \\ -A^T & D_2 \end{bmatrix}. \tag{2.23}$$

In Section 2.3.3.2, with $K = 2$, we show that the second eigenvector of the generalized eigenproblem $Lv = \lambda Dv$ provides a real relaxation to the discrete optimization of fining the minimum normalized cut. In [9], the author uses Expressions 2.22 and 2.23 in the generalized eigenproblem to find the solution. Let vector v be a vertical combination of vector v_1 and of vector v_2. The following equation can be derived:

$$\begin{bmatrix} D_1 & -A \\ -A^T & D_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \lambda \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}. \tag{2.24}$$

From Equation 2.24, it is easy to obtain:

$$D_1 v_1 - A v_2 = \lambda D_1 v_1, \quad -A^T v_1 + D_2 v_2 = \lambda D_2 v_2. \tag{2.25}$$

Assuming that each document contains at least one term and each term is contained at least in one document, D_1 and D_2 are then non-singular. Equation 2.25 can then be expressed as:

$$D_1^{1/2} v_1 - D_1^{-1/2} A v_2 = \lambda D_1^{1/2} v_1, \quad -D_2^{-1/2} A^T v_1 + D_2^{1/2} v_2 = \lambda D_2^{1/2} v_2. \tag{2.26}$$

Equation 2.26 can be further simplified into:

$$(1-\lambda) D_1^{1/2} v_1 = D_1^{-1/2} A v_2, \quad (1-\lambda) D_2^{1/2} v_2 = D_2^{-1/2} A^T v_1. \tag{2.27}$$

Let $\boldsymbol{\mu} = D_1^{1/2} v_1$ and $\boldsymbol{\nu} = D_2^{1/2} v_2$ hold, this produces $v_1 = D_1^{-1/2} \boldsymbol{\mu}$ and $v_2 = D_2^{-1/2} \boldsymbol{\nu}$. With these, Equation 2.27 can be rewritten as:

$$(1-\lambda)\boldsymbol{\mu} = D_1^{-1/2} A D_2^{-1/2} \boldsymbol{\nu}, \quad (1-\lambda)\boldsymbol{\nu} = D_2^{-1/2} A^T D_1^{-1/2} \boldsymbol{\mu}. \tag{2.28}$$

Let $A_n = D_1^{-1/2} A D_2^{-1/2}$. Accordingly, $A_n^T = D_2^{-1/2} A^T D_1^{-1/2}$ holds. Let $\sigma = (1-\lambda)$ Equation 2.28 becomes:

$$\sigma \boldsymbol{\mu} = A_n \boldsymbol{\nu}, \quad \sigma \boldsymbol{\nu} = A_n^T \boldsymbol{\mu}. \tag{2.29}$$

Equation 2.29 in fact represents the canonical forms of the Singular Value Decomposition (SVD) of matrix A_n. σ is a singular value, while $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ are the corresponding left and right singular vectors for σ. As shown in Section 2.3.3.2, when $K = 2$, the solution to Problem 2.17 is the eigenvector that associates to the second eigenvalue, λ_2. As $\sigma_2 = 1 - \lambda_2$, the solution to Equation 2.24 is then the left and right singular vectors $\boldsymbol{\mu}_2$ and $\boldsymbol{\nu}_2$ that associate to σ_2. $\boldsymbol{\mu}_2$ indicates the bi-partitioning of documents, while $\boldsymbol{\nu}_2$ gives the indicator of bi-partitioning for terms. The full computing procedure of this bi-partitioning approach is presented in Algorithm 6.

Though the mathematical reasoning of this approach starts from Expressions 2.22 and 2.23, in fact the final solution only depends on A_n, which is much smaller than the other matrices. This is an obvious advantage.

Based on Algorithm 6, when $K > 2$, a multi-partitioning solution can be obtained in a recursive manner. However, in [9], a more direct approach is provided by applying

Algorithm 6 Spectral Bi-partitioning on documents and terms [9]

Require: An input document-term matrix A of shape $n \times m$

1: Compute diagonal matrices D_1 and D_2 from A.
2: Compute matrix A_n by $A_n = D_1^{-1/2} A_n D_2^{-1/2}$.
3: Apply SVD on A_n to obtain the left and the right singular vectors $\boldsymbol{\mu}_2$ and $\boldsymbol{\nu}_2$ that associate to the second singular value σ_2.
4: Obtain the optimal eigenvector v in the generalized eigenproblem $Lv = \lambda Dv$ by computing $v_1 = D_1^{-1/2}\boldsymbol{\mu}_2$ and $v_2 = D_2^{-1/2}\boldsymbol{\nu}_2$.
5: For $i = 1, \ldots, n+m$, let $y_i \in \mathbb{R}$ be the vector corresponding to the i-th row of v.
6: Cluster the points $(y_i)_{i=1,\ldots,n+m}$ with the K-means algorithm into clusters C_1, C_2.

Ensure: Clusters V_1 and V_2 with $\{V_i = j | y_i \in C_i\}$.

K-means on a matrix that is constructed from singular vectors $\boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_{\lceil \log_2^K \rceil + 1}$ and $\boldsymbol{\nu}_2, \ldots, \boldsymbol{\nu}_{\lceil \log_2^K \rceil + 1}$. In fact, this matrix can be interpreted as projections of documents and terms in a space of $\mathbb{R}^K$. This is referred as "spectral embedding" [63]. Though there is nothing proved to use K-means clustering in this space to construct discrete partitions from the real valued representation y_i, the Euclidean distance turns out to be a meaningful quantity. This provides favorable evidence for applying K-means to achieve multi-partitioning in [9], where the author experimentally demonstrates good clustering quality. Since the Euclidean distance is considered as a meaningful metric in spectral embedding, and clustering using K-means in this space is experimentally proven to be good, analogously, it is reasonable to apply other clustering methods, such as hierarchical clustering methods, to obtain co-clusters organized in a hierarchy.

A drawback of [9] is that there is no discussion on the computing complexity of Algorithm 6. A similar co-clustering method is proposed in [67]. Likewise, it also uses the left and the right singular vectors to solve the graph partitioning problem, with the purpose to obtain balanced clusters by minimizing *Ncut*. The major difference this approach introduces is that it constructs multi-partitioning by recursively performing bi-partitioning. Besides, a discussion on computation complexity to obtain the left and the right singular vectors is elaborated. Zha et al. [67] state that the computation complexity of their approach is $O(CK_{svd}M)$, where C denotes the number of recursions, K_{svd} denotes the number of singular values and M denotes the number of nonzero values in matrix A_n. Usually, the computation complexity of a complete SVD process can reach $O(\min(nm^2, mn^2))$ in decomposing a matrix of size n-by-m [73]. However, in this work, the authors apply the Lanczos bi-diagonalization procedure [74] to reduce the complexity of obtaining the left and the right singular vectors. In principle, this procedure goes through a number of iterations to compute partial SVDs. In each iteration, two matrix-vector multiplications $A_n\boldsymbol{\mu}$ and $A_n^T\boldsymbol{\nu}$ are calculated. Consequently, the computation complexity of obtaining the left and the right singular vectors becomes proportional to the number of nonzero values in the input matrix A_n.

The approaches proposed in [9, 67] are sometimes referred to "Spectral-SVD" co-clustering methods. As these approaches are based on a relaxation imposed on the optimization problem (Section 2.3.3.2), there is no guarantee on the quality of such a solution with respect to the exact solution. An good example can be found in [75], where the authors prove that Spectral-SVD methods fail to produce the best partition in a cockroach graph, which looks like a ladder with several rimes removed. Figure 2.15 (a) illustrates the optimal cut on a cockroach graph obtained by either *RatioCut* or *Ncut*, which results in two horizontal sub-graphs. However, by Spectral-SVD, it obtains two vertical sub-graphs shown as Figure 2.15 (b), where the *cut* is not optimized.

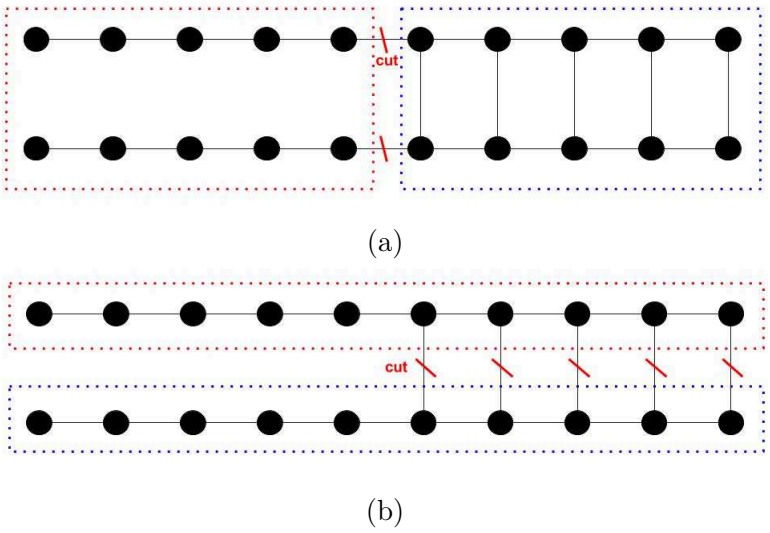


FIGURE 2.15: A cockroach graph partitioned by (a) an ideal cut and by (b) a Spectral-SVD method

To overcome this problem, Rege et al. [68] propose the Isoperimetric Co-clustering Algorithm (ICA), which is another approach that performs co-clustering by partitioning a weighted bipartite graph. However, different from the Spectral-SVD methods that try to minimize *Ncut* or *RatioCut*, ICA heuristically minimizes the ratio of the perimeter of the bipartite graph partition and the area of the partition under an appropriate definition of graph-theoretic area. In the experiments, ICA is proven to partition the cockroach graph more effectively than the Spectral-SVD methods. However, it is also more difficult to implement.

2.3.4 Co-clustering Using Non-negative Matrix Factorization

2.3.4.1 Non-negative Matrix Factorization

Apart from discovering co-clusters using graph-partitioning approaches, another important branch of methods is based on matrix factorization. As in text clustering, the input

TF-IDF matrices are usually non-negative, therefore non-negative matrix factorization (NMF) is much studied for clustering as well as for co-clustering.

Given a non-negative matrix $A = (a_{ij})_{i=1,\ldots,n;j=1,\ldots,m}$, NMF tries to decompose A into two non-negative matrices, such that $A \approx FG$, where $F \in \mathbb{R}^{n\times k}$ and $G \in \mathbb{R}^{m\times k}$. F and G can be obtained by solving:

$$\min \|A - FG^T\|^2 \; s.t \; F, G \geq 0. \tag{2.30}$$

As the function $\|A - FG^T\|^2$ is not convex with variables F and G, it is only possible to find local minima to the optimization problem. Lee and Seung [76] propose "multiplicative update rules" as a method to solve the optimization problem. They claim that it is a good compromise between speed and ease of implementation, in comparison with gradient descent and conjugate gradient methods. NMF can be applied in a wide range of domains, such as image processing, recommender systems and document clustering [77]. If A presents a document-term matrix, NMF provides F and G as the clustering indicator matrices of documents and of terms, respectively. In this case, there are k clusters of documents and k clusters of terms. In [78], it is shown that when the orthogonal constraint $G^TG = I$ is added, NMF becomes equivalent to K-means clustering.

2.3.4.2 Approaches for Co-clustering

Li and Ding [78] survey a range of NMF variations, among which *Tri-Factorization* (TF) based methods are often used to solve co-clustering problems. Briefly, TF aims to decompose matrix A into three non-negative matrices. A formulation of TF proposed in [79] is described as:

$$\min \|A - FSG^T\|^2 \; s.t \; F^TF = G^TG = I \; and \; F, S, G \geq 0$$

where $F \in \mathbb{R}^{n\times k}$, $S \in \mathbb{R}^{k\times c}$ and $G \in \mathbb{R}^{m\times c}$. Matrix S is introduced to absorb the different scales between A, F and G. Besides, it offers increased degrees of freedom, so that low-rank matrix representation remains accurate when F returns row clusters and G returns column clusters. An obvious drawback of TF is that it is computationally expensive, due to intensive matrix multiplications. Wang et al. [80] propose a Fast Nonnegative Matrix Tri-factorization (FNMTF) that reduces the computational cost by using fewer matrix multiplications. The objective function is defined as:

$$\min \|A - FSG^T\|^2 \; s.t. \; F \in \Psi^{n\times k}, \; G \in \Psi^{m\times c}.$$

The main difference between TF and FNMTF is that the orthonormal constraints on F and G change. The outputs of FNMTF are matrix F for row clustering and matrix G for column clustering.

2.3.4.3 Connection Between NMF and Spectral Graph Partitioning

Ding et al. [81] show an interesting link between NMF and graph partitioning via spectral clustering. They prove that using graph Laplacian matrix in minimizing the objective function such as *Ncut* and *RatioCut* can be equivalently carried out via non-negative matrix factorization. Concretely, this association is introduced by inserting a division of $vol(V)$ to the objective function in Expression 2.17. With $L = D - W$ and $f^T Df = vol(V)$, a new objective function can be written as:

$$\min \quad \frac{f^T(D-W)f}{f^T Df} \quad s.t. \quad Df \perp e. \tag{2.31}$$

Recall that $f = D^{-1/2}g$ is used to establish Expression 2.18. With $g = D^{1/2}f$, the authors use a normalized vector h, such that $h = \frac{g}{\|g\|}$, to rewrite Expression 2.31 as

$$\min \quad h^T(I - \tilde{W})h \quad s.t. \quad h \perp D^{1/2}e \tag{2.32}$$

with $\tilde{W} = D^{-1/2}WD^{-1/2}$. As the scaled cluster indicator vector h obeys the orthonormal condition, its matrix $H = (h_1, \ldots, h_n)$ satisfies $H^T H = I$. With $h^T Ih$ being a constant, minimizing the objective function 2.32 is equivalent to

$$\max \quad Tr(H^T \tilde{W} H) \quad s.t. \quad H^T H = I, \quad H > 0. \tag{2.33}$$

The authors point out that allowing H to be continuous is the spectral relaxation of *Ncut*. The solution to Expression 2.33 is given by the k principle eigenvectors of matrix $\tilde{W}$. This approach is previously detailed in Section 2.3.3.3. Another option the authors provide is to rewrite Expression 2.33 into

$$\min \|\tilde{W} - HH^T\|^2 \quad s.t. \quad H \geq 0. \tag{2.34}$$

Expression 2.34 is in the conventional form of NMF. It thus can be solved as an NMF problem. Once H is obtained, the original cluster indicator vector f can be obtained by optimizing

$$\min \quad \|h - \frac{D^{1/2}f}{\|D^{1/2}f\|}\|^2. \tag{2.35}$$

This gives an equivalent solution to using spectral graph partitioning.

2.3.5 Distributed and Parallel Approaches

Along with approaches that try to improve computational efficiency from algorithmic aspects, there exist many methods that take advantage of hardware parallelism. Papadimitriou and Sun [82] propose DisCo, a scalable framework under which co-clustering algorithms that employ a checkerboard structure can be implemented with MapReduce. Given a matrix A of n rows and m columns, and r and c denoting row group assignments and column group assignments, the goal of DisCo is to find r and c such that, after permutation based on r and c, the correlated sub-matrices are grouped together. As searching for the optimal group assignments is NP-hard, DisCo introduces a local search by alternating between row and column assignments while holding the other assignments fixed.

BiTM (Bi-clustering using Topological Maps) performs co-clustering based on self-organizing maps [83]. Its implementation is built on Apache Spark. BiTM consists of a topological map and a set of observations. It iteratively looks for the column and row assignments that associate observations to the topological map. The output is a block structure.

Su et al. [84] introduce sequential updates for alternate minimization co-clustering (AMCC) [54] and propose Co-ClusterD, a distributed framework that consists of two approaches to parallelize AMCC by (1) dividing clusters and (2) batching points. Co-ClusterD is implemented based on iMapReduce [85], a distributed framework on Hadoop that supports iterative algorithms.

2.4 Tests of the Cluster Hypothesis

2.4.1 Overview

Information retrieval (IR) is a wide domain, there are many research works that contribute in different directions. For example, collaborative IR analyses user-to-user collaboration in order to perform shared IR tasks [86, 87]; selective search provides perspectives on how to organize a very large document collection so that it can be searched accurately

and efficiently [88, 89]; query performance prediction is studied to estimate retrieval effectiveness in the absence of relevance judgments [90, 91]; and cluster-based IR systems apply clustering methods in performing IR tasks [92, 93, 94].

The cluster hypothesis is the fundamental assumption of applying clustering methods in IR. This hypothesis states that "the associations between documents convey information about the relevance of documents to requests" [92]. It implies that, for a query, relevant documents tend to be more similar to each other than the non-relevant documents. Therefore, relevant documents are likely to appear in the same clusters. For more than four decades, different tests on the cluster hypothesis have been experimented from various aspects. They provide theoretical yet valuable knowledge in studying the retrieval effectiveness and efficiency of a target clustering method in an IR application.

Based on objectives and approaches, these tests can be categorized into classic tests, refined tests and language-model based tests.

2.4.2 Classic Tests

Generally speaking, classic tests can be classified into two kinds. The first kind focuses on comparing retrieval effectiveness between a specifically designed cluster based search and conventional inverted file system (IFS) search. The overlap test [92], the nearest neighbor test [93] and the density test [95] are representatives of this kind. They first examine whether the cluster hypothesis characterizes the experimented collections, then compare the effectiveness of retrieving documents using a designed search strategy, against document-based search. The other kind of tests emphasizes a specific clustering family, hierarchical clustering, which is capable to explicitly reveal the connections of all documents, and is thus more informative and beneficial in retrieving documents. While many works compare retrieval effectiveness among a set of hierarchical clustering methods, some other tests examine the different strategies of searching documents in the dendrogram output by a hierarchical clustering method.

2.4.2.1 Comparison Tests of Cluster-based and Document-based Search

Overlap Test. Jardine and van Rijsbergen [92] test the cluster hypothesis based on the assumption that "given a query, the similarity between two relevant documents should be higher than the similarity between a relevant and a non-relevant document". They measure the overlap (or say the separation) between two similarity distributions of relevant pairs and non-relevant pairs. A collection with a low overlap value is believed to cluster strongly together for a set of queries, and relevant documents are thus better separated

from non-relevant documents. Within this experimental setting, their overlap test using the Cranfield-200 collection concludes that cluster-based searches have the potential to greatly outperform IFS. The same result is obtained in a later test, in which the full Cranfield[15] collection is experimented [96].

Nearest Neighbor Test. Voorhees [93] provides an alternative method, the nearest neighbor test. She assumes that the k nearest neighbors of a document d are the k documents that are the most similar to d. Therefore, if the cluster hypothesis holds for a collection, then many of the nearest neighbors of a relevant document would also be relevant. For each relevant document of each query, this test examines whether the assumption holds by computing the k nearest neighbors of a relevant document and recording the number of relevant documents in the set of neighbors. Four datasets (MED, CACM, CISI[16] and INSPEC) are experimented, with $k \in [0, 5]$. The nearest neighbor test concludes that the cluster hypothesis holds for the MED collection, but not for the CISI collection. The extent to which the cluster hypothesis characterizes a collection seems to have little effect on how well cluster searching performs as compared to a sequential search of the collection [93].

Density Test. El-Hamdouchi and Willett [95] propose the density test. The concept of *density* is defined as "the total number of postings in the collection divided by the product of the number of documents in the collection and the number of terms which have been used for the indexing the collection". For example, given a collection of n documents indexed by an average of m terms, which are selected from a vocabulary of M terms, the density is m/M. Given a collection, documents that only have a few terms from the vocabulary are unlikely have common terms with other documents. Thus they tend to have low inter-document similarities. However, when a document-term matrix is densely populated, documents tend to share a large number of terms, therefore higher similarities are expected. Based on this assumption, the density test expects that collections with high density values would give better results in clustered search than those with low values. This test is empirically demonstrated to be more correlated than the overlap and the nearest-neighbor tests, with the relative improvement posted by cluster-based retrieval [97]. Besides, this test does not require query or relevance data, so it can be applied to test collections whose query set is unavailable.

2.4.2.2 Tests Using Hierarchical Clustering

Comparing retrieval effectiveness among different hierarchical clustering methods.

[15] A textual collection of 1,400 short abstracts of aeronautical system articles.
[16] `http://ir.dcs.gla.ac.uk/resources/test_collections/`

Inspired by the initial work on the cluster hypothesis [92], many subsequent works devote their tests to examining the retrieval effectiveness of different hierarchical clustering methods. Differing from flat clustering, hierarchical clustering is more informative in revealing the internal connections among a set of input documents. It outputs a binary tree structure, named dendrogram, which is able to explicitly illustrate how individual documents are grouped. In cluster-based retrieval applications, dendrograms can illustrate how relevant documents are located, and thus can be very beneficial in guiding information seeking. In Section 2.2.1, we state that the agglomerative hierarchical clustering (AHC) is practically preferred to divisive hierarchical clustering, as the latter can be NP-hard. In testing the cluster hypothesis, many research works have extensively applied four AHC methods to compare their retrieval effectiveness. These methods are single link, complete link, average link and Ward method. However, in terms of retrieval effectiveness, results of these tests are not consistent. Griffiths et al. [98] claim that "average link gave the best results" in his test, while Willett [99] concludes that single link displays the poorest performance, and that "complete link is probably the most effective method". Yet, Griffiths et al. [100] state that "Ward's method was found to give the best overall results". In addition, discussion on computing efficiency is usually out of the scope of these works. As complexity of conventional AHC is up to $O(n^3)$, efficiency is an important factor to consider in practice. Moreover, a common incompleteness of these works is that they do not cover all members in the conventional AHC family. As illustrated in Figure 2.10, there are seven conventional AHC methods, but only four of them have been tested and compared.

Comparing cluster-based searching strategies.

Based on hierarchical clustering, another branch of tests concentrates on comparisons of two different cluster-based searching strategies: top-down search and bottom-up search [96, 99]. The result of these strategies is a cluster of documents instead of a set of ranked documents. In the top-down search, the query enters via the root of the dendrogram. It is then matched against two child clusters. The child cluster with higher query-cluster similarity is chosen to continue the search. The search moves down until it finds one cluster that satisfies a pre-defined retrieval criterion. This pre-defined retrieval criterion can be a minimal number of relevant documents contained in the cluster. Alternatively, it can be a critical similarity value, at which the similarities of query and clusters start to decrease. This searching strategy has a time complexity of $O(\log n)$. However, a general problem of this strategy is that it often produces a large cluster that contains most of relevant documents and many non-relevant documents, resulting in high recall and low precision. This problem can be avoided using the bottom-up search, which starts with leaf clusters of the dendrogram and moves up toward the root until some retrieval criterion is met. Nevertheless, this search is not efficient if there are too many

documents at the leaf nodes. A recommended way is to determine a starting cluster for the search by manually selecting a single relevant document at the leaf level. However, this search strategy is likely to output a relevant cluster that has high precision but low recall. In [99], it is stated that "bottom-up searching will generally give better results than top-down searching, especially when the very small bottom-level clusters are used". A similar conclusion is stated in [94], where the author proves that bottom-up search is superior to top-down search via a probabilistic model.

Optimal cluster search.

In both searching strategies, it is required to compute the similarity between a query and a cluster representative. Usually, a centroid is used as the representative of a cluster. When the size of a cluster is small, such a representative can be considered to be a reasonable description. However, for a large cluster, it becomes less suitable to represent a cluster using a representative. Differing from the two search strategies, the **optimal cluster search** [92] does not involve actual matching between a query and a cluster representative. In scanning the hierarchy of the dendrogram, a cluster is considered as the optimal cluster if it has the minimal E-measure, which is defined as

$$E = 1 - \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{2.36}$$

with P and R denoting the precision and recall, respectively. β is a parameter that balances the importance of precision and recall. It usually takes values 0.5, 1 and 2. When $\beta = 1$, the E-measure is equivalent to the F-measure.

Compared to top-down search and bottom-up search, optimal cluster search is superior, as it adopts the (harmonic) mean of precision and recall. It avoids to retrieve either large clusters that result in high precision and low recall, or small clusters that have high recall and low precision. In addition, it is still possible to adjust between precision and recall by choosing different values for β. Another advantage of optimal cluster search is that it does not require to compute any cluster representative. In fact, it directly concerns the internal connections of documents in the dendrogram when it computes the retrieval effectiveness for a query. Thus, it eliminates the bias brought from external sources to the document hierarchy.

2.4.3 Refined Tests

Several recent works refine some of the classic tests by adopting a different distance measure or new experimental settings.

Smucker and Allan [101] claim that the nearest neighbor test applies an insufficient measure in testing the cluster hypothesis, as it only requires documents to be locally clustered. Thus, it cannot distinguish whether a set of relevant documents are locally or globally clustered. For some similarity measures and some document collections, the nearest neighbor test may fail to detect when relevant documents do cluster well. By modeling documents into a weighted and directed graph, this work proposes a new measure, the normalized mean reciprocal distance (nMRD), which is able to capture global document-to-document similarities. It proves to be an effective measure for testing the cluster hypothesis.

Tombros [102] refines the overlap test and re-examines the retrieval effectiveness of hierarchical clustering in a dynamic fashion. Differing from the previous tests designed in [93, 94], in which clustering is applied statically over the whole document collection prior to querying. This test applies clustering only on the top-n ranked documents, which are retrieved in response to a query. The retrieval effectiveness is then measured by optimal cluster search, and compared against an IFS. The conclusions of such dynamic cluster-based retrieval confirm that the effectiveness of optimal query-specific cluster significantly outperforms optimal IFS. Moreover, under this dynamic setting, average link outperforms single link, complete link and Ward method.

2.4.4 Language Model-based Tests

All the tests mentioned above are based on the "bag-of-words" assumption and the vector space model. Another set of tests on the cluster hypothesis adopt the language model [103, 104]. In such a model, a document is not considered as a set of terms, but as a distribution of terms. A detailed survey on the language model can be found in [105].

Differing from vector space model-based tests, in which similarity/dissimilarity measures are often the Euclidean distance, the cosine similarity, the Dice coefficient, or the Jaccard coefficient. In language model-based tests, this metric is measured by the expected negative cross entropy between the documents' Dirichlet-smoothed unigram language models, i.e.,

$$S(d_i, d_j) = \exp\left(- CE\left(p_{d_i}^{Dir[0]}(.)||p_{d_j}^{Dir[\mu]}(.)\right)\right)$$

where d_i and d_j are two documents, CE is the cross entropy (or Kullback-Leibler divergence), Dir is the Dirichlet smoothing, and μ is its parameter. μ is set to 1000 in [104].

Raiber and Kurland [104] model several web-scaled datasets using the language model and retest the nearest neighbor test. They conclude that "the cluster hypothesis can hold, as determined by a specific test, for large scale Web corpora to the same extend it does for newswire corpora". In comparing the retrieval effectiveness of the nearest neighbor clusters, the single link hierarchical clusters and the document-based ranking system, they state that the nearest neighbor clusters outperform the others.

Raiber and Kurland [104] also retest the overlap test, the density test and the nMRD test on top-n retrieved documents ($n = 50, 100, 250, 500$) from nine various datasets [103]. The objective is to find the impact of correlation between a cluster hypothesis test and the effectiveness of cluster-based retrieval. Their work reveals that the correlation between the two is influenced by the type, the size of collections, the tested methods, and the number of documents in the retrieved list.

2.4.5 Applications of the Cluster Hypothesis in IR

The Cluster Hypothesis provides the core assumption and theoretical base for selective search, a parsimonious retrieval strategy in IR. Selective search partitions a collection based on document similarity in order to obtain topic-based subsets, and it searches only a few subsets that are estimated to contain relevant documents for a given query [88]. Compared to traditional search, which divides the collection into subsets and then processes the query against all shards in parallel, selective search outperforms its counterpart using limited computing resources. A well-studied question in selective search is how to smartly partition a collection so that documents in the same partitions are semantically similar.

Apart from academic research, commercial cluster-based retrieval systems provide alternatives to commonly-used document-based search engines. Yippy.com[17] and Carrot2.org[18] are two search engines of this genre. In responding to a query, a list of ranked documents and a list of concept clusters are returned. By clicking a cluster label, a set of child concept clusters displays. The ranked documents change accordingly, as users click the clusters that they are interested in. Compared to document-based search, this searching procedure is more navigating and helpful when user's information need is unclear.

[17] `https://yippy.com/`
[18] `http://search.carrot2.org/stable/search`

2.5 Conclusion

In this chapter, we aim to present the state of the art that is related to the content of this thesis.

Firstly we introduce the basic knowledge on text clustering, distributed and parallel computing. Then, we emphasize on three topics: agglomerative hierarchical clustering, co-clustering and tests on the cluster hypothesis. Detailed concepts and existing approaches are presented for each topic. Concretely, in agglomerative hierarchical clustering, we introduce approaches that are most related to the methods concerned in this thesis. They are the conventional AHC methods and the Lance-Williams formula. Besides, the nearest neighbor chain approach is also presented and discussed.

With respect to co-clustering, we first present a statistical model, the latent block model. Then, we detail the graph partitioning approach by elaborating on its properties and mathematical insight, which offer good explanations to one of the proposed methods in this thesis, the SHCoClust. Moreover, we introduce another co-clustering approach, which is based on non-negative matrix factorization. As this method is related to the Spectral-SVD methods, we believe it offers a good insight for the proposed method.

Comes lastly is the tests on the cluster hypothesis. We detail three main kinds of tests, the classic tests, the refined tests, and the language model-based tests. By reading this section, reads can have a good understanding on the concept of the cluster hypothesis, as well on different tests proposed in the past. In this thesis, we contribute to the classic tests. Concretely, we propose new tests that apply Sim_AHC and SHCoClust, with focus on comparing retrieval effectiveness among different hierarchical clustering methods and addressing efficiency issue.

Chapter 3

The Similarity-based Agglomerative Hierarchical Clustering Framework

3.1 Motivation

As stated in Chapter 1, we are interested in agglomerative hierarchical clustering (AHC), as it is capable to explicitly reveal the internal connections of data instances. In this sense, it is superior to flat clustering. In addition, AHC is more efficient than its counterpart, the divisive hierarchical clustering, which can reach NP-hard. However, as mentioned in section 2.2, chapter 2, AHC methods are still computationally costly. The time complexity of computing a conventional AHC method can reach $O(N^3)$. Nearest neighbor chain (NN-chain) methods are more efficient [33, 106], as their time complexity is up to $O(N^2)$. Yet, as they are constrained by the reducibility property, these methods cannot work with median and centroid methods. Others methods, such ad CURE [38] and BIRCH [6], claim to address large datasets. But they also have disadvantages. For example, CURE reduces data by random sampling and partitioning. Though it decreases time complexity to $O(N^2_{sample} log N_{sample})$, the results are indeterministic due to the random procedures. As to BIRCH, its time complexity is $O(N)$, but an extra structure, the clustering features (CF) tree, has to be employed in order to store compact summaries of the original data. A more recent approach, SparseHC [41], is capable to perform online AHC by principally structuring clusters with an adjacency hash map. According to experimental results, this method does decrease memory growth, but time complexity is improved only for single link. Besides, this approach is not generic, it is only applicable for single link, complete link and average link.

We are interested in a method that is generic for all conventional AHC methods, independent from any external structure, deterministic in its results and scalable. Scalability

is an important property for an algorithm. In parallel and distributed computing, a scalable algorithm means that computing can be performed concurrently, so that the total running time decreases as the number of computing units increases on the same input. In a traditional computing mode, where computing is performed sequentially, a scalable algorithm can be interpreted as being capable to perform the same computing task while occupying much less resources, such as memory and running time, compared to a non-scalable algorithm on the same input. Therefore, such an algorithm can process larger datasets with limited computing resources available.

To satisfy our requirements, we propose a similarity-based agglomerative hierarchical clustering framework, Sim_AHC. It is called “a framework" as it can address all of conventional AHC methods. It is thus *generic*. In addition, it does not require any other external structure, nor a random sampling procedure. Therefore, its results are *deterministic* and it is *independent* from any external structure. Sim_AHC can be considered as a new expression of the Lance-Williams (LW) formula, but uses inner product-based similarities instead of distances. This change provides two important advantages.

- On one hand, as the similarities lie between 0 and 1, it is feasible to employ a filtering strategy to sparsify the input similarity. This results in less memory to store non-zero similarity values, as well as less computing time to perform clustering. Sim_AHC is thus *scalable*.
- On the other hand, as similarities are based on inner products, it is possible to use different kernel functions to compute similarities. This property allows Sim_AHC to perform AHC methods on non-linearly separable datasets more effectively.

This chapter is organized as follows: in Section 3.2, we detail Sim_AHC from several aspects: the mathematical deduction, its extension to kernel functions, and some advice on sparsifying the input similarity matrix. Through the mathematical deduction, the readers can have a good understanding on how we replace distances in the Lance-Williams formula with inner product similarities to produce Sim_AHC. Besides, detailed reasoning on the equivalence of the two frameworks is illustrated as well. We then show how the basic form of inner product similarity can be extended to any kernel function. Next, we point out how to correctly sparsify a general similarity matrix, which can be obtained from negative vectors. After the theoretical properties of Sim_AHC are explained, we present our experiments and results in Section 3.3. Our experiments using linear kernel and Gaussian kernel aim at testing (1) the equivalence between Sim_AHC and its counterpart, the Lance-Williams formula, and (2) the impact of sparsifying the input similarity matrix on scalability and on clustering quality. Our empirical results tested on three datasets demonstrate that Sim_AHC is equivalent to the AHC algorithm using

the Lance-Williams formula. In addition, the impact of sparsifying the input similarity matrix is positive, for the gain of efficiency does not compromise clustering quality. Concretely, clustering quality tends to be preserved as the input similarity matrix is getting more and more sparsified. It collapses when the matrix is largely sparsified at some level. Before this collapsing point, up to 90% memory usage can be freed, and up to 85% running time is saved with our tested datasets. Meanwhile, clustering quality is preserved as high as or better than using full-sized similarity matrix as input.

3.2 The Similarity-based Hierarchical Clustering Framework, Sim_AHC

In this section, we introduce Sim_AHC from several aspects. Firstly, we start with the mathematical reasoning to show how this framework is deduced and how it is equivalent to the Lance-Williams formula. Secondly, we extend the framework to kernel functions. Lastly, we present a strategy to sparsify the input similarity matrix.

3.2.1 Mathematical Deduction

Assuming we are dealing with a dataset of n instances and m features, let a matrix A of shape n-by-m represent this dataset. With x and y being two row vectors from A, the Euclidean distance of x and y can be expressed as:

$$D_{Euclidean}(x, y) = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}. \tag{3.1}$$

The inner product of x and y is defined as:

$$\langle x, y \rangle = \sum_{i=1}^{m} x_i y_i. \tag{3.2}$$

It is clear that the Euclidean distance of x and y can be expressed in the form of an inner product, so as the squared Euclidean distance. Using inner products, the squared Euclidean distance of x and y can be expressed as:

$$D^2_{Euclidean} = \langle x, x \rangle + \langle y, y \rangle - 2\langle x, y \rangle. \tag{3.3}$$

Let us assume that the similarity between x and y, $S(x, y)$, is defined by the inner product of their normalized form, i.e.,

$$S(x, y) = \langle \frac{x}{||x||}, \frac{y}{||y||} \rangle. \tag{3.4}$$

$S(x, x) = S(y, y) = 1$ always holds. Given Equation 3.3, the corresponding dissimilarity $D(x, y)$ can be defined as:

$$D(x, y) = ||\frac{x}{||x||} - \frac{y}{||y||}||^2 = S(x, x) + S(y, y) - 2S(x, y) = 2(1 - S(x, y)). \tag{3.5}$$

Now let x and y be two vectors C_x and C_y, each representing a cluster. Under the above assumptions, we can provide an equivalent expression of the Lance-Williams formula using S instead of D. Recall that in the first iteration of the AHC algorithm (Algorithm 1), there are n clusters, each being one data point C_x. With Equation 3.5, we can deduce a relationship shown as follows:

$$\begin{aligned}
(C_i, C_j) &= \arg\min_{(C_x, C_y)} D(C_x, C_y) \\
&= \arg\max_{(C_x, C_y)} \big(-\frac{1}{2} D(C_x, C_y) \big) \\
&= \arg\max_{(C_x, C_y)} \Big[S(C_x, C_y) - \frac{1}{2} \big(S(C_x, C_x) + S(C_y, C_y) \big) \Big].
\end{aligned} \tag{3.6}$$

When the pair (C_i, C_j) that has the largest similarity is found, C_i and C_j are merged into one cluster $C_{(ij)}$. With C_k denoting any remaining cluster, for the subsequent iterations, we show that the Lance-Williams formula can be recast as follows:

$$-\frac{1}{2} D(C_{(ij)}, C_k) = S(C_{(ij)}, C_k) - \frac{1}{2} \big(S(C_{(ij)}, C_{(ij)}) + (\alpha_i + \alpha_j) S(C_k, C_k) \big) \tag{3.7}$$

where:

$$\begin{aligned}
S(C_{(ij)}, C_k) &= \alpha_i S(C_i, C_k) + \alpha_j S(C_j, C_k) + \beta S(C_i, C_j) \qquad (3.8) \\
&\quad -\gamma |S(C_i, C_i)/2 - S(C_i, C_k) - S(C_j, C_j)/2 + S(C_j, C_k)| \\
S(C_{(ij)}, C_{(ij)}) &= (\alpha_i + \beta) S(C_i, C_i) + (\alpha_j + \beta) S(C_j, C_j). \qquad (3.9)
\end{aligned}$$

Formulas 3.6 and 3.7 are equivalent to $(C_i, C_j) = \arg\min_{(C_x, C_y)} D(C_x, C_y)$ in the conventional AHC procedure, while the recurrence formulas 3.8 and 3.9 are the counterparts of the Lance-Williams formula (Equation 2.1) that establishes our method. With $S(C_x, C_x) = 1$ holding for all n singletons $\{C_x\}$, we show below that our formulation is equivalent to the Lance-Williams formula for each clustering scheme listed in Table 2.5:

- **Single link and complete link methods.**

  It is easy to show that Equation 3.8 reduces to $S(C_{(ij)}, C_k) = \alpha_i S(C_i, C_k) + \alpha_j S(C_j, C_k) + \beta S(C_i, C_j) - \gamma |S(C_i, C_k) - S(C_j, C_k)|$, and Equation 3.9 reduces to $S(C_{(ij)}, C_{(ij)}) = 1$, as $\alpha_i + \alpha_j + 2\beta = 1$ with parameter values in Table 2.5. Since $\alpha_i + \alpha_j = 1$ holds in Equation 3.7, then Equation 3.6 with the reduced updating rules of Equations 3.8 and 3.9 are globally equivalent to the conventional procedure. Note that, for the other methods, $\gamma = 0$, so Equation 3.8 boils down to $S(C_{(ij)}, C_k) = \alpha_i S(C_i, C_k) + \alpha_j S(C_j, C_k) + \beta S(C_i, C_j)$.

- **Average link, McQuitty, centroid, median and Ward methods.**

  Recall that $S(C_x, C_x) = 1$ holds for all n singletons $\{C_x\}$. In Equation 3.7, when the term $S(C_{ij}, C_{ij})$ is replaced by Equation 3.9, with some simple linear algebra manipulations, Equation 3.7 becomes:

  $$-\frac{1}{2} D(C_{(ij)}, C_k) = S(C_{(ij)}, C_k) - (\alpha_i + \alpha_j + \beta).$$

  With the value of the term $-(\alpha_i + \alpha_j + \beta)$ being equal or greater than -1, we can divide the five clustering methods into two groups: (1) the average link, McQuitty and Ward methods that have $-(\alpha_i + \alpha_j + \beta) = -1$, and (2) the centroid and median methods who satisfy $-(\alpha_i + \alpha_j + \beta) > -1$:

  1. Regarding the average link, McQuitty and Ward methods, with the aid of Table 2.5, it is not difficult to prove that $-(\alpha_i + \alpha_j + \beta) = -1$ always holds. Consequently, for these methods, Equation 3.6 with the updating rules $S(C_{(ij)}, C_k) = \alpha_i S(C_i, C_k) + \alpha_j S(C_j, C_k) + \beta S(C_i, C_j)$ and $S(C_{(ij)}, C_{(ij)}) = 1$ is globally equivalent to the general AHC procedure.
  2. Concerning the centroid and median methods, since $\alpha_i + \alpha_j = 1$ in Equation 3.7, the coefficient assigned to $S(C_k, C_k)$ vanishes. However, $\alpha_i + \alpha_j + 2\beta \neq 1$ in Equation 3.9. Hence, $S(C_{(ij)}, C_{(ij)}) \neq 1$. Therefore, it is important to apply the weighting system determined in Equation 3.9 for the global equivalence of the centroid and median methods to hold.

Wrapping up all particular cases discussed above, the computing procedure of Sim_AHC goes through the following steps.

- At each iteration, we solve:

$$(C_i, C_j) = \underset{(C_x, C_y)}{\arg\max}\, S(C_x, C_y) - \frac{1}{2}\big(S(C_x, C_x) + S(C_y, C_y)\big). \tag{3.10}$$

- After having merged (C_i, C_j) into $C_{(ij)}$, the similarity matrix S is updated by applying the two following equations:

$$\begin{aligned} S(C_{(ij)}, C_k) &= \alpha_i S(C_i, C_k) + \alpha_j S(C_j, C_k) + \beta S(C_i, C_j) \\ &\quad -\gamma |S(C_i, C_k) - S(C_j, C_k)| \end{aligned} \tag{3.11}$$

$$S(C_{(ij)}, C_{(ij)}) = \delta_i S(C_i, C_i) + \delta_j S(C_j, C_j). \tag{3.12}$$

The new expression leaves the values of parameters α_i, α_j, β and γ unchanged as in the original Lance-Williams formula. To guarantee the equivalence for each clustering method, the values of the newly added parameters δ_i and δ_j can be determined freely, as long as their sum satisfies the following conditions:

$$\delta_i + \delta_j = \begin{cases} \frac{1}{2} & \text{for median} \\ \frac{|C_i|^2 + |C_j|^2}{(|C_i| + |C_j|)^2} & \text{for centroid} \\ 1 & \text{for other methods.} \end{cases}$$

At beginning of this Section, we use A to denote a data matrix. In a text mining task, $A = \{a_{(ij)}\}_{i=1,\ldots,n;j=1,\ldots,m}$ can be considered as a document-term matrix of shape n-by-m, with n denoting the number of documents in the corpus. After some preprocessing, each document is represented by a vector of m terms. When the TF-IDF weighting strategy is applied, a_{ij} is the TF-IDF value of the j^{th} term in the i^{th} document. Before input A to Sim_AHC, it is important to perform row-wise normalization on A, so that each individual document vector is scaled to unit norm. Let $a_{i.}$ denote a document vector represented by m TF-IDF values. It is suitable to adopt the l_2-norm to normalized $a_{i.}$ into:

$$\hat{a_{i.}} = \frac{a_{i.}}{\|a_{i.}\|} \quad with \quad \|a_{i.}\| = \sqrt{\sum_{j=1}^{m} a_{ij}^2} \tag{3.13}$$

with $\|a_{i.}\|$ denoting the norm of $a_{i.}$ and $\hat{a_{i.}}$ denoting the normalized form. Normalization on A outputs $\hat{A}$. The input to Sim_AHC is the pairwise similarity matrix S of $\hat{A}$, $S = \hat{A}\hat{A}^T$. Algorithm 7 details the computing procedure of Sim_AHC.

Algorithm 7 Sim_AHC computing procedure

Data: The pairwise similarity matrix S

Initialize a dendrogram of n leaves with null height values

while *number of iterations* $< n$ **do**

1. Search for the pair of clusters (C_i, C_j) that has the maximal similarity in S, by Equation 3.10,
2. Merge C_i and C_j into $C_{(ij)}$ and add a corresponding parent node in the dendrogram with height value $[S(C_i, C_j) - \frac{1}{2}(S(C_i, C_i) + S(C_j, C_j))]$,
3. Compute the similarity of $C_{(ij)}$ and any other cluster C_k and update S accordingly, by Equations 3.11 and 3.12.

end

Result: A dendrogram of $2n - 1$ nodes

Unlike the dendrogram output by a conventional AHC method, the dendrogram output by Sim_AHC is growing downwards. The height of a newly merged cluster is $[S(C_i, C_j) - \frac{1}{2}(S(C_i, C_i) + S(C_j, C_j))]$. This is equal to $-\frac{1}{2}D(C_i, C_j)$, a negative value.

3.2.2 Extension to Kernel Functions

Recall that the basic assumption of Sim_AHC is Equation 3.4. The usage of inner products in Sim_AHC allows us to naturally extend its clustering methods to kernel functions [107]. Consequently, in our method, broader similarity measures can be easily employed and non-linearly separable datasets can be addressed more effectively.

Let $\phi : \mathcal{I} \to \mathcal{F}$ represents a mapping from a low dimensional space to a possibly infinite dimensional space. A kernel function K on two data vectors x and y in space $\mathcal{I}$ is defined as $K(x, y) = \langle \theta(x), \theta(y) \rangle$. It is in fact the inner product of two mapped vectors in space $\mathcal{F}$. If K is a linear kernel, the S matrix in our approach is filled with cosine similarities, and more importantly, its diagonal entries should be constant. Gaussian and Laplacian kernels satisfy this condition naturally, but other kernels must be normalized. To generalize all cases, we obtain a normalized kernel similarity matrix by:

$$S(x, y) = \frac{K(x, y)}{\sqrt{K(x, x)K(y, y)}}. \tag{3.14}$$

3.2.3 Sparsification of the Cosine Similarity Matrix

For a document-term matrix A that is filled with TF-IDF values, as TF-IDF values are non-negative, the similarity values in the resulting similarity matrix S lie between 0 and 1. However, in the general case, S can contain negative values. In that case, let $m < 0$ be the minimal value in S and $|m|$ its absolute value. It is always possible to transform S in order to have non negative values using the following re-scaling operator, $\forall x, y \in \hat{A}$:

$$S(x, y) \leftarrow \frac{S(x, y) + |m|}{1 + |m|}. \tag{3.15}$$

Since this mapping is monotonically increasing, S remains an inner product matrix and, in addition, its diagonal is filled with value 1.

Assuming that S is non negative, we propose to apply a simple thresholding operator that depends on a parameter $\tau \in [0, 1]$. Any pair of clusters that has a similarity value below τ is considered being far enough from each other, thus their similarity is replaced by 0. Similarities that are over τ are unchanged in S. Formally, this process can be expressed as:

$$S(x, y) \leftarrow S(x, y) I_{(S(x,y) \geq \tau)} \tag{3.16}$$

where $I_{(S(x,y) \geq \tau)} = 1$ if $S(x, y) \geq \tau$ and $I_{(S(x,y) \geq \tau)} = 0$ otherwise.

When S exists in memory as a sparse matrix, i.e., only non-zero values are actually stored, the thresholding strategy with τ results in an S matrix sparser than the original one. Therefore, less memory is required to store S and less computing time is demanded to compute on it. However, this strategy is unsuitable for distances, even normalized distances that have value between 0 and 1. Because for distances, zero and close-to-zero values are the most important as they signify high similarity, these values have to be stored instead of being ignored.

Next, we propose to restrict the search for pairs of clusters to merge in Equation 3.10 into the following subset: $\mathbb{S} = \{(C_k, C_l) : S(C_k, C_l) > 0\}$. This permits to reduce running time, since the bottleneck of the general AHC algorithm is actually the time used in searching for the optimal proximity value, which has $O(N^2)$ time complexity. Accordingly, we propose to replace Equation 3.10 with:

$$(C_i, C_j) = \underset{(C_k, C_l) \in \mathbb{S}}{\arg\max}\, S(C_k, C_l) - \frac{1}{2}(S(C_k, C_k) + S(C_l, C_l)). \tag{3.17}$$

As we shall see in the next section, this approach not only dramatically reduces processing time, but also achieves better clustering results.

3.3 Experimental Verification

The goals of our experiments are to demonstrate that:

1. under the assumptions exposed previously, the framework based on Equations 3.10, 3.11 and 3.12 is equivalent to the conventional AHC procedure (Algorithm 1), and the Lance-Williams formula (Equation 2.1);

2. sparsifying the cosine similarity matrix obtained by Formula 3.16, and applying Sim_AHC (Equations 3.17, 3.11 and 3.12) considerably decreases memory use and running time, while better clustering results can be obtained.

To this end, we experiment on text clustering tasks. Indeed, hierarchical clustering is particularly interesting in this case, since it allows expressing the relationships between different topics in a collection and at different granularity levels. Moreover, cosine similarities are classic proximity functions used for documents. In addition, hierarchical document organization based on the conventional AHC procedure faces the problem of scalability, since text collections are usually large. Our experiments seek to overcome these limits.

It is important to note that our purpose is not to compare different AHC methods among each other, but rather to exemplify the properties of Sim_AHC in comparison to the conventional AHC procedure that use the Lance-Williams formula. As a consequence, the results obtained by the conventional approach serve as our baseline.

3.3.1 Datasets, Preprocessing and Evaluation Measures

In our experiments, we use three well-known corpora that are employed in text clustering benchmarks: Reuters-21578[1] [108], SMART [9] and 20Newsgroups[2] [109]. The Reuters corpus is a collection of newswire articles, assembled and indexed with categories by personnel from Reuters Ltd. Its ApteMode is a collection of 10,788 financial articles from 90 categories (classes). This collections is skewed (yet less than the original corpus) with 36.7% of the documents in the most common classes, and only 2 documents in each of the five least common classes. In our experiment, we generate a sample of the ApteMode collection, named Reuters, by firstly ignoring the classes that have less than

[1] Distribution 1.0, ApteMod version.
[2] We use the same dataset as in `http://qwone.com/~jason/20Newsgroups/`.

50 documents, and then randomly sampling (without replacement) a number of articles from each remaining class. The SMART corpus contains three classes, MENDLINE, CISI and CARNFILED, containing 1033, 1460, and 1400 journal abstracts on medicine, informatics and aerodynamics, respectively. The 20Newsgroups corpus contains 18,821 news articles from 20 classes. The number of documents in each class varies between 628 to 999. In our experiments, we use a sample of the 20Newsgroup corpus, named 20NG, by randomly selecting (without replacement) 300 documents from 15 classes. Details of datasets used in our experiments can be found in Table 3.1.

Dataset	Classes included	No. Cla	No. Docs	No. Terms
Reuters	earn×1000, acq×700, crude×150, money-fx×150, grain×200, interest×100, trade×100, ship×50, wheat×50, corn×50	10	2446	2547
SMART	MEDLINE×1033, CISI×1460, CRANFILED×1400	3	3893	3025
20NG	misc.forsale, sci.electronics, comp.sys.ibm.pc.hardware, rec.sport.hockey, talk.politics.guns, comp.os.ms-windows.misc, soc.religion.christian, rec.autos, rec.motorcycles, sci.crypt, sci.med, sci.space rec.sport.baseball, talk.politics.mideast, comp.graphics	15	4483	4455

TABLE 3.1: Descriptions of experimented datasets

Based on the "bag-of-words" assumption and the vector space model, a simple preprocessing is performed on each dataset: (1) terms that appear in less than 0.2% and in more than 95% documents in a collection are removed, (2) no stemming or lemmatization is applied on the remaining terms, no stop word is removed, (3) the TF-IDF weighting scheme is applied, and (4) l_2 normalization on each document vector is applied [3]. "No.Docs" and "No.Terms" in Table 3.1 indicate the number of documents and the number of terms after preprocessing. Consequently, preprocessing generates a document-term matrix, where each row is a document vector represented by a set of terms in the columns.

We use the adjusted Rand index (ARI) and (the absolute value of) the cophenetic correlation (CC) between dendrograms to compare the clustering outputs. CC is employed to evaluate how far our dendrogram is from the one produced by the conventional AHC procedure. In this case, higher is better and the maximum value 1 means that the dendrograms are equivalent and thus represent the same hierarchy. ARI is an external assessment criterion that evaluates the quality of the clustering output in regard to a given ground-truth. It requires to flatten the dendrogram with the correct number of clusters, then the obtained partition and the ground-truth are compared to each other.

[3] A Python script for preprocessing is available at `https://github.com/xywang/text_preprocessing/blob/master/preprocessing.py`

Greater ARI values imply better clustering outputs. The maximum value 1 is observed when the ground-truth is perfectly recovered.

3.3.2 Experiment Settings and Results

Given a document-term matrix A, two types of matrices are generated: the cosine similarity matrix S and the corresponding distance matrix D as defined by Equation 3.5. Note that since the document-term matrix consists of non negative values, similarity values in S are all between 0 and 1 ($s \in [0, 1]$), therefore no rescaling operator is needed.

Given a clustering method, the S matrix is taken as input to Sim_AHC, while the related dense matrix D is taken as input to the conventional AHC algorithm. Consequently, two dendrograms are returned and we compute the CC in order to assess the similarity between the two outputs. Two cases are of interest: (1) when $\tau = 0$, which means no sparsification and the dense S is used; and (2) when $\tau > 0$ and increases, which leads to sparser and sparser S matrices.

In addition to 0, we choose other threshold values τ at the 10^{th}, 25^{th}, 50^{th}, 75^{th} and 90^{th} percentiles of distribution of values in S. Let k denote the rank of a percentile so that $k \in \{0, 10, 25, 50, 75, 90\}$ with the convention that the 0^{th} percentile is 0. Accordingly, when k grows the k^{th} percentile τ is greater and greater and the S matrix becomes sparser and sparser.

We experiment with two types of kernel: linear and Gaussian. The linear kernel is simply the inner product in $\mathcal{I}$ between normalized vectors as defined in Equation 3.4. The Gaussian kernel between two points $x, y \in \hat{A}$ is given by $K(x, y) = \exp(-\gamma \|x - y\|^2)$. It corresponds to a cosine measure in $\mathcal{F}$. In our experiment we set γ to $1/p$ by default[4].

In Figure 3.1, we show the results obtained for all seven methods on the Reuters, SMART and 20NG datasets, respectively. Results for linear and Gaussian kernels are arranged in the left and right blocks. Rows correspond to AHC methods (using their abbreviations) and columns to collections. In each graph, each point corresponds to one of the measurements listed afterwards with respect to an S matrix; the x-axis corresponds to percentile ranks (divided by 100) which define the threshold values τ (not shown); solid lines with plus signs represent the relative memory use; dashed lines with cross signs show the relative running time; and dotted lines with circle symbols indicates the absolute value of cophenetic coefficient (CC), dotted lines with triangle symbols give the ARI values. We report the curves of several measurements (y-axis) when S is progressively sparsified as the percentile rank (x-axis) increases. In addition to CC and ARI graphs (dotted lines

[4]Note that this default setting is used in popular SVM packages. In this case γ is very low, the Gaussian kernel provides values close to 1 and data points in a pair is close to each other.

with circle and triangle symbols, respectively), the percentage of memory use of a sparse S with respect to the dense S, i.e., $\frac{memory\,use\,at\,\tau_i \neq 0}{memory\,use\,at\,\tau=0}$, and the proportion of running time when using a sparse S as compared to the dense S, i.e., $\frac{running\,time\,at\,\tau_i \neq 0}{running\,time\,at\,\tau=0}$, are plotted as well (solid lines with plus symbols and dashed lines with cross symbols, respectively). Therefore, memory and processing time costs related to the full S (corresponding to the 0^{th} percentile where $\tau = 0$) serve as baselines (with y-axis value of 100%). In these cases, the lower the percentages the bigger the gains.

3.3.2.1 Equivalence between Sim_AHC and the Lance-Williams Formula

In Figure 3.1, for all datasets and both kernels, CC values are all equal to one when $\tau = 0$ (0^{th} percentile shown at the origin). This empirically demonstrates that our approach is equivalent to AHC using the Lance-Williams formula.

Next, as percentile rank increases, CC values generally decrease, illustrating the fact that dendrograms move away from Lance-Williams formula based results. However, when using the linear kernel, CC values generally remain high even when the majority of similarity values are removed. Concerning the Gaussian kernel, CC values drop rapidly after having thresholded 10% of the lowest similarities, but they start increasing again after this fall.

However, the single link method presents a peculiar behavior: for all collections and both kernels, it always recovers the result given by the usual AHC procedure despite the fact that 90% of the S matrix is sparsified. In other words, our framework is able to obtain the same dendrogram as the original Lance-Williams formula, but with 90% of memory usage and running time saved.

3.3.2.2 Impact of Sparsifying Similarities on Scalability

Let $M \leq N^2$ be the number of non-zero cells in S. The storage cost of our approach is $O(M)$. The time complexity[5] is $O(NM)$, which indicates a linearly relationship with respect to storage complexity.

In Figure 3.1, solid lines with plus symbols give the percentage of size of the sparse S with respect to the dense S. As expected, this quantity linearly decreases as the percentile rank k grows.

Next, dashed lines with cross signs show the proportion of processing time observed with a sparse S with respect to the running time achieved with the dense S. We observe

[5]Similarly to the general AHC algorithm based on a dissimilarity matrix where $M = N(N-1)$.

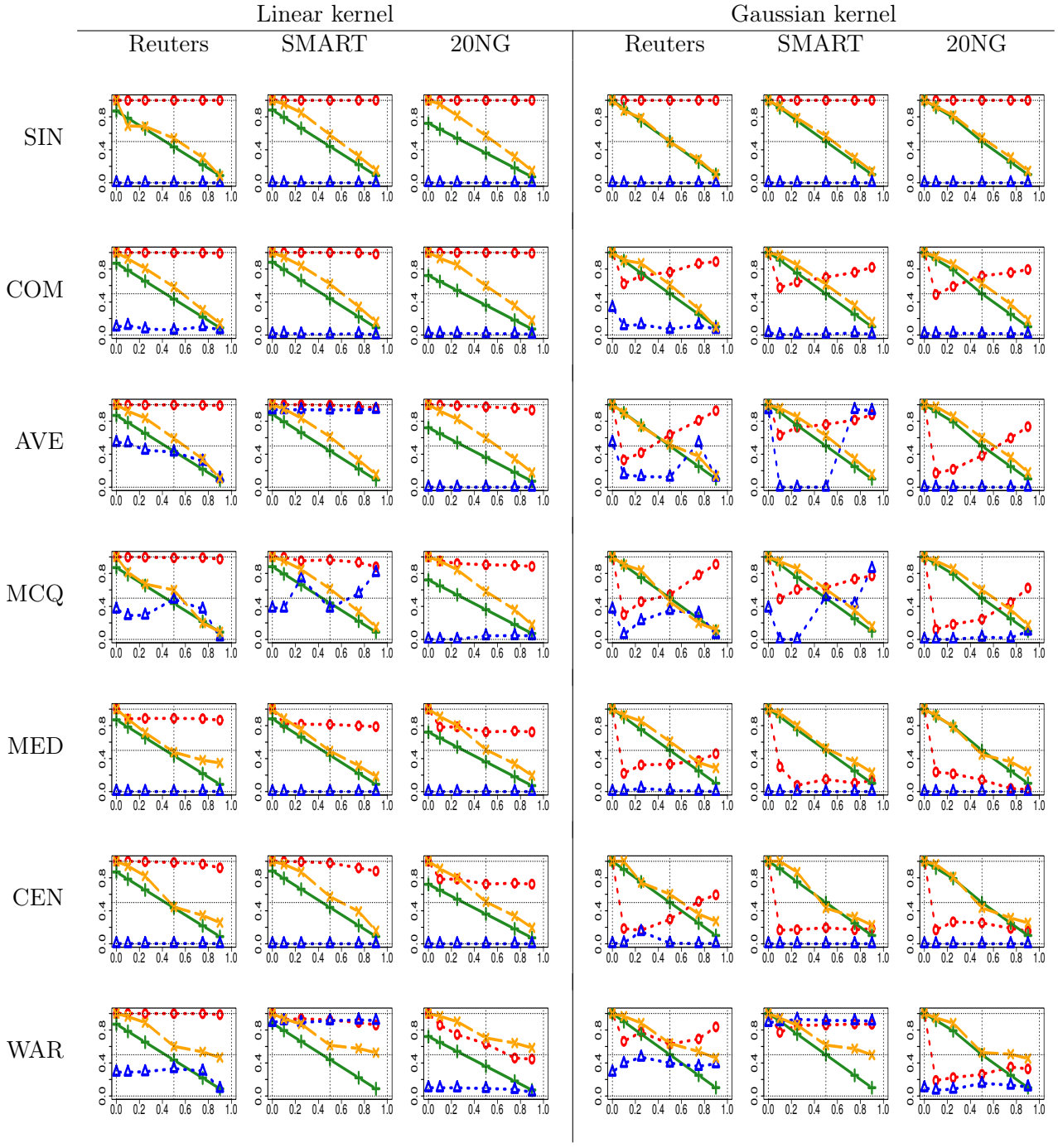


FIGURE 3.1: Results of applying Sim_AHC on the Reuters, SMART and 20NG datasets using linear and Gaussian kernels

linear curves as well, which depicts the linear relationship between memory and time complexities.

The sparsification of the S matrix enables decreasing storage complexity and running time. Besides, it also has an impact on clustering quality.

Previously, we have noticed that CC values were decreasing as S was sparser and sparser. In the following, we examine some cases in which our framework wins on both sides: scalability and quality.

3.3.2.3 Impact of Sparsifying Similarities on Clustering Quality

We focus on the quality of clustering output by analyzing ARI values. We observe that average link, McQuitty and Ward techniques work out better in general. Surprisingly, many of the best results are obtained with a very sparse S matrix and not with the full one. In Table 3.2, we report the best outcomes of this phenomenon. Mem% and Time% indicate the percentage of saved memory and processing time, respectively, when the sparse S is compared to the dense S.

	Method	kernel	τ	Mem%	Time%	CC	ARI
Reuters	Average	Gaussian	0	0	0	1	**0.543**
	Average	Gaussian	0.99	-75	-62	0.81	0.539
SMART	Average	Linear	0	0	0	1	0.939
	Average	Linear	0.078	-90	-85	0.96	**0.944**
20NG	Ward	Gaussian	0	0	0	1	0.100
	Ward	Gaussian	0.99	-50	-47	0.26	**0.154**

TABLE 3.2: Best ARI results for each collection when $\tau = 0$ (baseline) and when $\tau > 0$ (sparsified S)

For Reuters, the best ARI value is provided by average link with a full S given by the Gaussian kernel. However, comparable performance is obtained with the same method and kernel, but with a sparse S that saves 75% memory and 62% processing time.

Concerning SMART, average link gives the best ARI value as well, but with a linear kernel. Compared to the Lance-Williams formula-based AHC algorithm, Sim_AHC obtains a higher ARI while saving 90% memory and 85% running time.

Regarding 20NG, the Ward technique with Gaussian kernel works out the best. Our method allows increasing the baseline ARI value up to 54% while consuming around half of memory and running time.

3.4 Discussion

We have introduced Sim_AHC, an equivalent formulation of the Lance-Williams formula based on cosine similarities instead of Euclidean distances. Our AHC procedure, which relies on this formulation and a sparsified cosine similarity matrix, not only has better scalability properties, but is also able to output better clustering results.

We believe that two reasons account for this phenomenon. Firstly, sparsifying the S matrix reduces the noise by removing the lowest similarity values, therefore leading to better clustering performances. Secondly, when two clusters (C_i, C_j) are merged together, their respective neighborhoods (clusters having a non null similarity value with C_i and C_j respectively) are fused as well, so that $C_{(ij)}$ has a larger neighborhood than both C_i and C_j. Furthermore, the updating rule (Equation 3.11) allows reinforcing the similarity value of $C_{(ij)}$ with C_k if the latter cluster belongs to both initial neighborhoods. In fact, our approach can be viewed as a sort of "transitive closure" starting with reliable seeds (the pairs with highest similarity values) and propagating similarities through "trusted" neighborhoods.

However, the main drawback of our method is that, either sparsifying S does not improve the ARI value at all (see complete link applied to Reuters with Gaussian kernel in Figure 3.1, for instance), or improvements are not regular and setting the threshold value τ becomes difficult. Further theoretical investigations should be undertaken in these respects to have a better understanding of the properties of our framework.

3.5 Conclusion

In this chapter, we introduce Sim_AHC, a similarity-based agglomerative hierarchical clustering framework. Unlike other similar methods proposed in the past, Sim_AHC is a generic framework as it covers all conventional AHC methods unified in the Lance-Williams formula. Its results are deterministic, because it does not involve any sampling process. It is also independent from any extra structure. Through mathematical reasoning and experimental verification on several text clustering tasks, we demonstrate that Sim_AHC is theoretically and empirically equivalent to the Lance-Williams formula. However, Sim_AHC can be easily extended to kernel functions, as it uses inner product at base. This features makes it capable to address non-linearly separable datasets more effectively, unlike the Lance-Williams formula. More importantly, having all similarities between 0 and 1 in this framework helps set up a threshold value to sparsify the input similarity matrix, in order to reduce memory use and running time. Our experiments show that up to 85% running time and 90% memory use can be spared using Sim_AHC.

Surprisingly, the efficiency gain does not compromise clustering quality, which tends to remain constant or even improve compared to using a full-sized input. This discovery proves that Sim_AHC is scalable, as it is capable to obtain the same or better results while using limited computing resources.

In the next chapter, we introduce a variation of Sim_AHC that is capable to group both documents and terms.

Chapter 4

The Similarity-based Hierarchical Co-clustering Method

4.1 Motivation

Introduced in Section 2.3 of Chapter 2, co-clustering is a subspace clustering method that simultaneously groups data instances and their features. It results in a number of co-clusters, each of which contains a partition of closely situated data instances and a partition of highly associated features. Depending on the content of an input dataset, data instances and their features coexist in the same co-cluster interpret each other accordingly. For example, performing co-clustering on a gene expression dataset, where genes are featured by a set of conditions, produces co-clusters, each containing a subgroup of genes that exhibit highly correlated activities for a subgroup of conditions [50]; for a textual dataset, where documents are featured by a set of terms, co-clustering can find "describing" terms for similar documents in a co-cluster.

We believe that this advantageous property of co-clustering provides interesting insights. For instance, when applying in cluster-based IR tasks, co-clustering can retrieves not only documents but also terms, providing useful extra knowledge in seeking information. However, knowledge on how co-clusters and how elements in a co-cluster are connected is absent. We are interested in a method that is able to organize co-clusters and elements in a co-cluster, illustrating their interconnections. This is our initial motivation to work on a hierarchical co-clustering method.

Previously, in 2.3 of Chapter 2, we review a number of co-clustering techniques. In principle, three groups of methods are presented: LBMs, graph partitioning approaches and NMF-based approaches. Comparing these methods, we choose to build a hierarchical

co-clustering method using Spectral-SVD, which is applied in some of the graph partitioning approaches. Our reasons are: first of all, Spectral-SVD-based graph partitioning approaches are more likely to be combined with a hard clustering method. However, to LBMs and NMF-based approaches, embedding an external clustering method in them is not trivial. Secondly, our basic assumption is the “bag-of-words" assumption and the vector space model, we consider a document as a set of terms instead of a distribution. Thus it is not very suitable to choose LBMs, which are dependent on the distribution of the input. LBMs are more suitable for the assumption of language model, where document is considered as a distribution of terms. Thirdly, compared to NMF-based approaches, the graph partitioning approaches that apply Spectral-SVD are based on a convex optimization problem. Though it relaxes some constraints, this optimization problem returns a global minima. However, NMF-based approaches are based on a non-convex optimization formulation, which only returns a local minima.

In this chapter, we introduce SHCoClust, a similarity-based hierarchical co-clustering method. Concretely, given a document collection as input, we consider it as a bipartite graph with documents and extracted terms being vertices. Firstly we apply “spectral embedding" to project the corresponding document-term matrix into a space, which is constructed by the eigenvectors of the graph Laplacian; then we apply Sim_AHC on the projected data to generate a dendrogram, which iteratively aggregates documents and terms; lastly, we cut this dendrogram to obtain a number of sub-dendrograms, each organizing a number of documents and a number of terms in a hierarchy. In the rest of this chapter, we present the computing procedure of SHCoClust in Section 4.2, and we empirically illustrate its properties by performing and evaluating several text clustering experiments in Section 4.3.

4.2 The Computing Procedure of SHCoClust

In Section 2.3.3.2, we illustrate mathematical details on how the discrete optimization problem of *Ncut* is relaxed when then we only need to cut a graph into two sub-graphs, $K = 2$. And we show that the relaxed optimization problem in fact has the form of the standard Rayleigh-Ritz theorem, Equation 2.18, to which the second eigenvector of the symmetric normalized graph Laplacian L_{sym} is the solution. Using this knowledge, we explain the Spectral Bi-partitioning Co-clustering method [9] in Section 2.3.3.3, it is shown that the solution can also be provided by the second left and right singular vectors of the diagonally normalized input document-term matrix, A_n, Equations 2.29. In fact, based on the relation, $\sigma_2 = 1 - \lambda_2$, where λ_2 indicates the second eigenvalue of L_{sym}, and σ_2 denotes the second singular value of A_n, the second eigenvector of L_{sym}

and the second left and right singular vectors of A_n provide the equivalent solution to the relaxed optimization problem of *Ncut*. However, compared to looking for the second left and right singular vectors of A_n, solving the problem using the second eigenvector of L_{sym} requires to perform computation on a much larger matrix, see Equation 2.23. In order to achieve better performance, we prefer to look for the singular vectors of A_n.

For multi-partitioning, where several sub-graphs (or clusters) are generated, there are two options: either recursively performing bi-partitioning, or applying a flat clustering method, such as K-means, on the resulted matrix composed by singular vectors. Comparing the two options, the second one is preferred, as recursively performing bi-partitioning is more computationally expensive than applying a flat clustering method. Moreover, the second option provides possibility to apply any other clustering method rather than K-means.

In the multi-partitioning case of Algorithm 6, applying K-means method returns flat co-clusters, which are of the same level. It is thus not possible to know how co-clusters are connected. In addition, inside such a co-cluster, the documents and terms are also of the same level, there is no information on how they are linked. Differently, AHC or Sim_AHC outputs a dendrogram. This is a binary tree structure that can display how documents are merged step by step. However, the connection among terms is ignored. It would be very interesting to have a hybrid structure that is capable to return co-clusters, and to display the connections of clustered documents and terms in a co-cluster, as well as to preserve the information on how co-clusters are linked at different levels. We believe that this hybrid structure that combines the properties of co-clustering and hierarchical clustering is more beneficial than any the contributing method. It surely provides us a better understanding of our data.

For such purpose, we propose SHCoClust, a similarity-based hierarchical co-clustering method. It is inspired by the Spectral Bi-partitioning Co-clustering algorithm [9], in the sense that it applies “spectral embedding" to project the input data into a space constructed by the singular vectors, then it performs clustering on the projected data in order to obtain real cluster labels for the input. Differently, we perform Sim_AHC on the projected data with the purpose to organize co-clusters and elements in an individual co-cluster in a form of dendrogram. The principle reason of choosing Sim_AHC over the conventional AHC method is that we are interested to make advantage of its scalability property. In addition, its compatibility with kernel functions makes it more favored over the conventional method.

The input to SHCoClust is a square data matrix. In the scope of this thesis, we address text clustering tasks, thus our input is a document-term matrix, denoted by A. Let n and m denote the number of rows and the number of columns of A, they also signify the

number of documents and the number of terms. The computing procedure of SHCoClust is presented in Algorithm 8.

Algorithm 8 Computing procedure of SHCoClust

Require: An input document-term matrix A of shape $n \times m$

1: Compute diagonal matrices D_1 and D_2 from A.
2: Compute matrix A_n by $A_n = D_1^{-1/2} \times A \times D_2^{-1/2}$.
3: Apply SVD on A_n to obtain l left singular vectors, $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_l$, and l right singular vectors $\boldsymbol{\nu}_1, \boldsymbol{\nu}_2, \ldots, \boldsymbol{\nu}_l$.
4: Construct matrix U by $\boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_l$, and matrix V by $\boldsymbol{\nu}_2, \ldots, \boldsymbol{\nu}_l$.
5: Apply matrix multiplication $D_1^{-1/2} \times U$ and $D_2^{-1/2} \times V$ to output Z_1 and Z_2.
6: Vertically combine Z_1 and Z_2 to output matrix Z.
7: For $i = 1, \ldots, n + m$, let $y_i \in \mathbb{R}$ be the vector corresponding to the i-th row of Z.
8: Apply l_2 normalization on Z, and compute pairwise similarity matrix S from it.
9: Re-scale S by Formula 3.15.
10: **while** num_iterations$< n$ **do**
11: Search for the pair of rows (Z_i, Z_j) that has the maximal similarity in S, by Equation 3.10,
12: Merge Z_i and Z_j into $Z_{(ij)}$ and add a corresponding parent node in the dendrogram with height value $[S(Z_i, Z_j) - \frac{1}{2}(S(Z_i, Z_i) + S(Z_j, Z_j))]$,
13: Compute the similarity of $Z_{(ij)}$ and any other row Z_k and update S accordingly, by Equations 3.11 and 3.12.
14: **end while**

Ensure: A dendrogram of $2(n + m) - 1$ nodes

The computing procedure of SHCoClust shown in Algorithm 8 is basically composed of two parts: spectral embedding that projects the input data into a space of eigenvectors, and application of Sim_AHC on the projected data to perform clustering.

1. Lines 1-6 correspond to the spectral embedding. This part is consisted of three steps: generation of the diagonally normalized matrix A_n, application of SVD and production of the projected data matrix Z.

   - **The generation of the diagonally normalized matrix A_n.** Given the document-term matrix A, we compute the diagonal matrices D_1 and D_2 by $D_1(i,i) = \sum_{j=1}^{m} A_{ij}$ and $D_2(j,j) = \sum_{i}^{n} A_{ij}$. D_1 and D_2 store the row sums and column sums of A. If A is represented as a graph, its documents and terms are vertices, its edges carry the TF-IDF weights, D_1 and D_2 are in fact the degree matrices for the document vertices and for the term vertices, respectively. Having D_1 and D_2, they generate A_n by matrix multiplication. The mathematics behind this is explained by Equations 2.25 to 2.29.
   - **The application of SVD.** Applying SVD on A_n with a given parameter l, the number of desired singular values, outputs l left singular vectors,

$\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_l$, and l right singular vectors $\boldsymbol{\nu}_1, \boldsymbol{\nu}_2, \ldots, \boldsymbol{\nu}_l$. A left singular vector $\boldsymbol{\mu}$ contains n elements, and a right singular vector $\boldsymbol{\nu}$ contains m elements. It is important to note that their corresponding singular values should be in such an order: $1 = \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_l$. Recall the relation between a singular value of A_n and an eigenvalue of the graph Laplacian L, $\sigma = 1 - \lambda$, and the eigenvalues are ordered as $0 = \lambda_1 \leq \lambda_2 \leq \ldots$. According to the Rayleigh-Ritz theorem, in a bi-partitioning task when there are two eigenvalues, the eigenvector that corresponds to the non-zero eigenvalue is the solution to the optimization Problem 2.18. When it comes to a multi-partitioning task, we are interested in the eigenvectors that correspond to the non-zero eigenvalues. In terms of singular vectors, those who correspond to singular values that are smaller than 1 provide the solution. This is why we exclude the first left singular vector and the first right singular vector in matrices U and V. U is of shape $n \times (l-1)$, and V is of shape $m \times (l-1)$. In [9], the author determines l by $l = \lceil \log_2 k \rceil$, where k is the number of desired clusters. However, it is lack of empirical training or theoretical proof. In our experiment, we learn this parameter by grid search for each tested dataset, and the l value that leads to the smallest difference between the original data and the projected data is selected.

- **The production of the projected matrix Z.** After U and V are generated, they are used to multiply D_1 and D_2, the degree matrices of document vertices and of term vertices. The process of multiplication can be considered as projecting documents to a space constructed by left singular vectors in U, and projecting terms to a space constructed by right singular vectors in V. Two matrices are output, Z_1 and Z_2, they contain projected documents and terms in the space of singular vectors. They are of shape $n \times (l-1)$ and $m \times (l-1)$, respectively. The following step is to vertically concatenate Z_1 and Z_2, that is, to fuse projected documents and terms in a space of $l-1$ dimensions. Performing clustering in this space outputs co-clusters consisted of documents and terms.

2. Lines 7-14, performing Sim_AHC on matrix Z to generate hierarchical co-clusters.

   As Z is a matrix of projected data mixed with documents and terms, it is important to index Z so that we are able to know which rows correspond to the set of documents and which rows correspond to the set of terms. Before preforming Sim_AHC, Z should be normalized so that each row has norm 1, this is based on the basic assumption of Sim_AHC, see Equation 3.4. Then we can compute the pairwise similarity matrix S from the normalized Z. If inner product is chose to compute the pairwise similarities, S is filled with cosine similarity values. And its

diagonal is constantly 1. Like in a common Sim_AHC process, kernel functions can be used to compute pairwise similarities, as shown in Equation 3.14. In our experiments, we use linear kernel (inner product) and Gaussian kernel to test SHCoClust in several text clustering tasks. In order to take advantage of the scalability property of Sim_AHC, we are interested to sparsify S to improve clustering efficiency. However, different from a document-term matrix filled with non-negative TF-IDF values, Z possibly contain many negative values, it is thus important to re-scale its pairwise similarity matrix S by Formula 3.15 in order to have all similarity values between 0 and 1. After the re-scaling, we are able to set up a threshold value $\tau \in [0, 1]$ to sparsify S in a proper manner.

Sim_AHC goes through a number of iterations. More S is sparsified, less iterations Sim_AHC needs to go through. In each iteration, a pair of document-document, term-term or document-term is merged and an internal node is created in the dendrogram. Due to the relation shown in Equation 3.7, the dendrogram grows downwards, the height of a newly merged node is negative, and its value is $[S(Z_i, Z_j) - \frac{1}{2}(S(Z_i, Z_i) + S(Z_j, Z_j))]$. A full dendrogram output by SHCoClust contains $n + m$ leaves and $n + m - 1$ internal nodes. Cutting the dendrogram at some height results in a number of flat co-clusters, each contains a sub-dendrogram of documents and terms, in Section 4.4, visualization of a sampled dataset is presented.

4.3 Experiments: Clustering Effectiveness and Efficiency

Our experiments on SHCoClust emphasize two aspects, the clustering effectiveness and the clustering efficiency. For the first aspect, we compare clustering quality among SHCoClust, the Spectral Bipartite Co-clustering method (SBC) [9] and the conventional AHC methods. In order to examine whether sparsification influences the clustering quality, we sparsify the similarity matrix S in SHCoClust and compare the clustering quality against the use case of using a full-sized S. As to clustering effectiveness, we are interested to examine how clustering quality is affected when S is getting more and more sparsified, using both linear kernel and Gaussian kernel.

4.3.1 Datasets, Preprocessing and Evaluation Measures

Six experimented datasets are sampled from the corpora mentioned in Section 3.3.1, their details can be found in Table 4.1. In the rest of the paper, theses datasets are referred by their indexes shown in column “Ind.". Based on the bag-of-words assumption

and the vector space model, each dataset is preprocessed into a document-term matrix, whose rows are document vectors and columns are terms. A few steps are involved in preprocessing, in detail, stop words are removed, terms that have document frequency higher than 20% and lower than 1% are removed, and TF-IDF weighting strategy is applied. “nb.docs" and “nb.terms" indicate the numbers of documents and of terms after the preprocessing. “K" is the ground-truth number of clusters, “l" is the learned parameter for SVD. “Z shape" indicates the shape of matrix Z obtained by the step of spectral embedding in Algorithm 8. “ARPACK"[1] is used as solver in SVD.

Dataset	Ind.	K	nb.docs	nb.terms	l	Z shape
Reuters	R5	5	500	652	4	(1152, 3)
	R7	7	2100	2133	4	(4233, 3)
	R10	10	2450	5075	5	(7525, 4)
SMART	S0	3	1500	2272	3	(3772, 2)
	S1	3	3893	6812	3	(10705, 2)
20NG	NG8	8	3200	1118	4	(4318, 3)
	NG20	20	2000	1104	6	(3104, 5)

Table 4.1: Experimented Datasets

To evaluate clustering quality, we use the adjusted Rand Index (ARI) [110] and the normalized Mutual Information (NMI) [111]. They both measure the similarity between a list of ground-truth cluster labels and a list of predicted labels. The Rand Index measures the similarity between two list of labels by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the two lists. The ARI is the “corrected" version of the Rand Index, because it is adjusted for chance using the scheme of: $ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}$, with RI stands for the Rand Index, and $E[.]$ denotes an expected value. The value of ARI ranges from 0 to 1, with 1 indicating that the two lists of cluster labels are identical, and 0 for being entirely different. NMI is normalization of the Mutual Information score. Its value is also between 0 and 1, with 1 indicating a perfect correlation between two compared lists, and 0 for non mutual information between the two. In order to use these two measures, we flatten the dendrogram obtained by SHCoClust to K clusters, then compute ARI and NMI using the predicted and the ground-truth cluster labels.

4.3.2 Comparisons of Clustering Effectiveness

We are interested in how well SHCoClust performs. Is it better or worse than the benchmark methods, such as SBC and the conventional AHC? And does sparsification affect clustering quality, positively or negatively? To answer these questions, we design our first experiments that are consisted of four tests as follows:

[1] `https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/utils/arpack.py`

1. SHCoClust without sparsification v.s. conventional AHC.
2. SHCoClust with sparsification v.s. without.
3. SHCoClust without sparsification v.s. BSC.
4. SHCoClust with sparsification v.s. BSC.

In tests that concern sparsification, threshold values τ at percentiles ranks {10, 25, 50, 75, 90}% of similarities in S are used to sparsify S. In this experiment, we apply linear kernel to obtain S, which is in fact filled with cosine similarities. The results of this experiment are illustrated in Figure 4.1. From top to down, each row maps to one test that is distinguished by its index. Each column lists the corresponding results for a tested dataset. Graph in each cell is a bar chart that is highlighted by a vertical line at x=0. The y-axis in each graph is labeled by the abbreviations of seven clustering methods. Bars in these graphs indicate the "difference" of values for ARI (the red, or the dark bars in a gray-scale printed paper) and for NMI (the blue, or the light ones) using a clustering method, obtained by subtraction of two tested approaches (A v.s. B, $A - B$). In the case of sparsification, we chose the highest value of ARI or NMI obtained in each clustering methods through the sequence of τ, then subtract it with the ARI or NMI value obtained by the approach in comparison.

In Figure 4.1, we can discovery a few interesting findings:

- Compared to conventional AHC, clustering quality is largely improved using SHCoClust for all clustering methods, except for single link and the Ward method. In datasets of S0 and S1, the increase is tremendous, ARI and NMI are raised up to 0.8 and 0.7 at maximum.
- When sparsification is applied in SHCoClust, the clustering quality is further enhanced. However, this enhancement does not compromise SHCoClust's efficiency. Instead, as most highest ARI and NMI values are obtained when $\tau \sim 1$, the efficiency of SHCoClust is considerably improved, thanks to a largely sparsified input.
- SHCoClust (without sparsification) obtains close results to SBC for all clustering methods, except for single link. Though improvements can be found in R5, NG8 and R7, other datasets do not display any. However, when sparsification is applied in SHCoClust, some noticeable amelioration can be observed. Comparing graphs of test 3) and of test 4), bars generally exhibit a left-to-right drift, diminishing their heights in the left side of the vertical line at x=0, and growing their heights in the right side. This drift is very apparent in R7 and NG20, where ARI and NMI

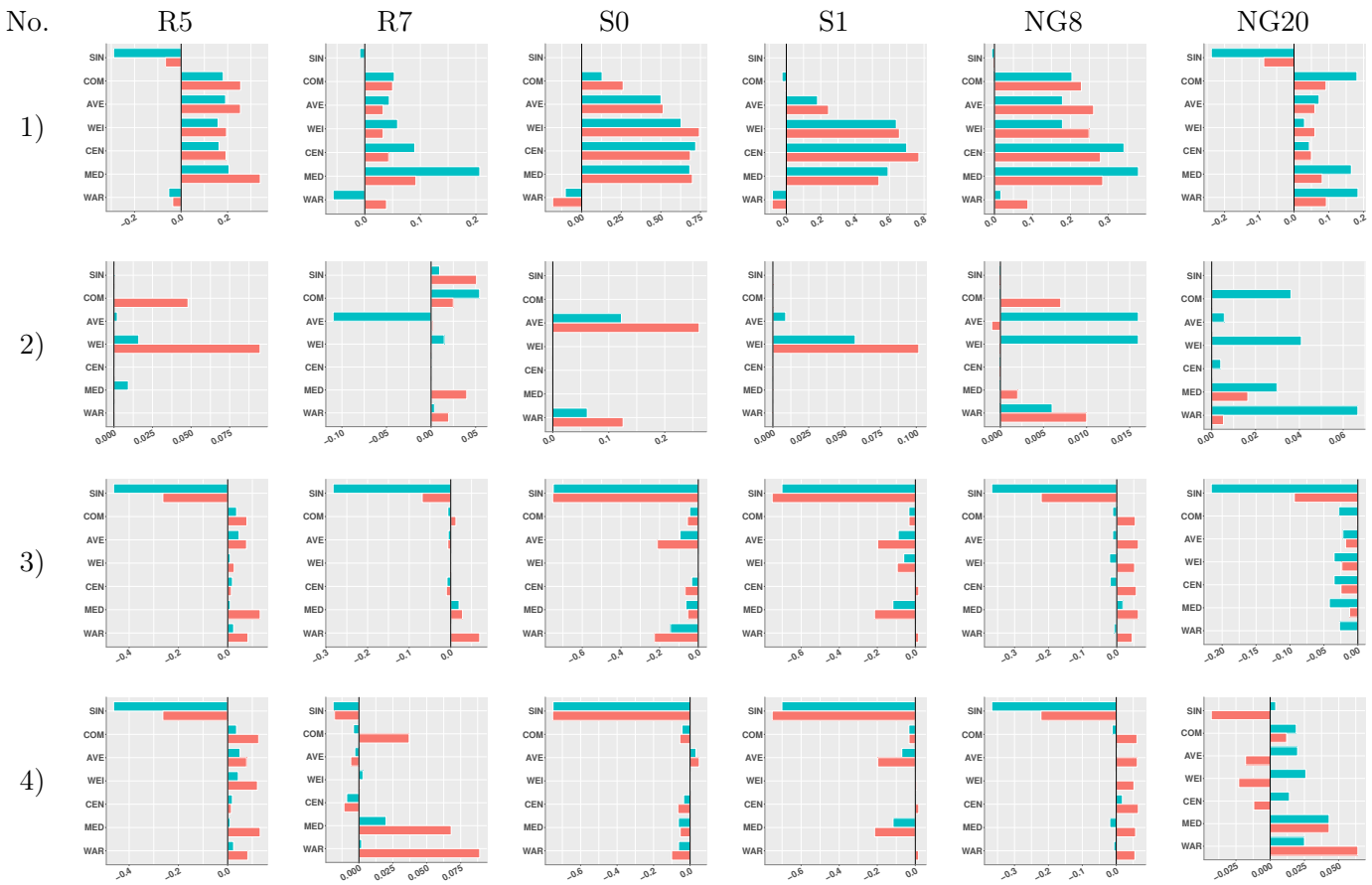


FIGURE 4.1: Comparisons of clustering quality among conventional AHC, BSC, SHCoClust with and without sparsification

bars are largely pulled towards right, regarding all clustering methods. However, in other datasets, single link seems to be the least affected.

To summarize, SHCoClust significantly improves clustering quality compared to the conventional AHC for most clustering methods. It obtains close results as SBC without sparsification. However, when sparsification is applied, SHCoClust demonstrates improvements over SBC.

4.3.3 Examination of Clustering Efficiency with Sparsification

In our second experiment, we examine the improvement of clustering efficiency when the similarity matrix S is getting more and more sparsified, and we record the changes of clustering effectiveness in order to observe if clustering quality compromises when computing efficiency is improved via sparsification. In addition, we extend the similarity function in SHCoClust to Gaussian kernel, given by $\mathcal{K}(x, y) = \exp(-\gamma\|x - y\|^2)$ for

$x, y \in \mathcal{D}$, $\gamma = 1/dim(\mathcal{I})$ by default[2]. Three representative datasets R5, S0 and NG8 are experimented and are compared to their results obtained by linear kernel. The input kernel matrix is sparsified in the same fashion as in the previous experiment. For each threshold value, ARI, NMI, relative memory usage and relative running time are recorded. Figure 4.2 illustrates the results of this experiment. Indexed by the abbreviations of clustering methods in row and the names of datasets in column, each graph in Figure 4.2 contains four lines: the solid line with "+" sign denotes the NMI values, the dashed line with triangle sign is for ARI, while the solid line with "o" sign and the solid line with "x" sign represent the relative memory usage and the relative running time, respectively. The x-axis corresponds to the percentile ranks that define the threshold values τ (not shown). At each $\tau \in (0, 1)$, exact memory usage and running time are recorded, and are divided by those at $\tau = 0$, i.e., when no thresholding is applied, to obtain the relative memory usage and the relative running time.

In Figure 4.2 we can see that, as the percentile rank increases, memory usage and running time decrease correspondingly, however, ARI and NMI tend to preserve their values at some level until the percentile rank approaches closely to 1. More precisely, before the percentile rank crosses 75%, in most cases, we can obtain ARI and NMI values as high as (or higher than) using the full-sized input. In some other cases, ARI and NMI even boost after percentile rank is over 75%, like in S0 (of linear kernel at average link). Among seven clustering methods, single link is the most peculiar, its ARI and NMI are invariant to the effect of sparsification, however, in R5 (of linear and Gaussian kernel) and NG8 (of Gaussian kernel), ARI of single link boosts at percentile rank = 90%, at which point memory usage and running time are largely reduced. Comparing the two kernel functions, overall similar behavior can be observed regarding the four curves. In some cases, Gaussian kernel returns higher metric values than linear kernel, and vice versa. Among the three experimented datasets, results of S0 are globally better, while R5 and NG8 display limited difference in their results.

Principle conclusions we can draw from this experiment are that:

- SHCoClust is capable to guarantee the clustering quality even when its input is largely sparsified, requiring considerably reduced memory usage and running time. τ at percentile rank of 75% is a heuristically good threshold in our experiment (90% for single link).
- The clustering quality does not compromise when computing efficiency is improved by sparsifying the similarity matrix S. On the contrary, clustering effectiveness is even increased when using a sparsified S. This is likely due to the fact that

[2]This default setting is used in popular SVM packages. Besides, when γ is low, Gaussian kernel provides close-to-one values and higher similarities between pairs of points.

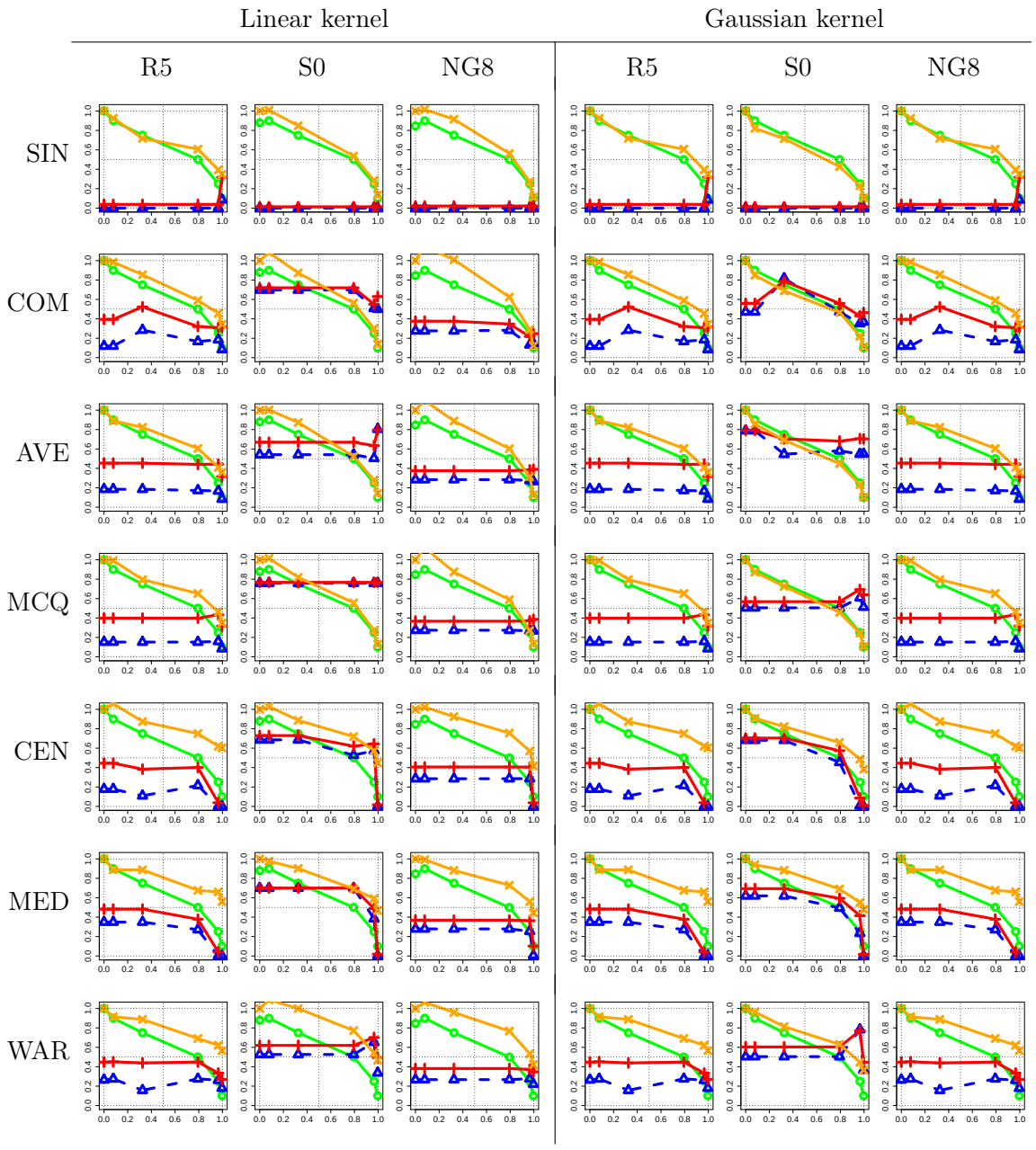


FIGURE 4.2: Results of linear kernel and Gaussian kernel with sparsification

sparsifying S matrix reduces the noise by removing the lowest similarity values, therefore leading to better clustering performances.

- SHCoClust can be easily employed with a kernel function, though only linear and Gaussian kernels are experimented here, we believe that its compatibility with kernel functions enables it to handle non-linearly separable datasets more effectively in other tasks.

4.3.4 Discussion of Complexity and Scalability

Mentioned in Section 2.3.3.3, the computation complexity of a complete SVD process can reach $O(\min(nm^2, mn^2))$ in decomposing a matrix of size n-by-m [73]. This is the complexity of performing spectral embedding in Algorithm 8. As matrix Z is of shape $(n+m)\times(l-1)$, the storage complexity for its pairwise similarity matrix S is $O((n+m)^2)$ and the time complexity of applying Sim_AHC on Z can reach $O((n+m)^3)$. Sparsifying S with τ results in M non-zero values stored for computation, $M < (n+m)^2$, eventually reduces storage complexity to $O(M)$ and time complexity to $O((n+m)M)$. The linear relationship between the storage and time complexity is demonstrated by the lines of relative memory and relative running time in Figure 4.2. Let τ^* define a threshold value, at which clustering quality is preserved as high as possible, meanwhile S is sparsified to have M as small as possible. As stated previously, τ at percentile rank of 75% is the τ^* for most cases in our experiment. Table 4.2 exhibits the consummation of resources measured by memory gain and time gain at τ^*, at which highest ARIs are obtained (marked by *). The clustering methods that output ARIs* and the ARI values of SBC are listed. $-x$ in Mem% and Time% indicates the relatively reduced memory and time compared to using a full-sized input.

Dataset	Method	τ^* value	Mem%	Time%	ARI*	BSC
R5	Median	0.242	-25	-8	0.395	0.263
R7	Ward	0.998	-75	-61	0.158	0.069
S0	Average	0.996	-90	-86	0.803	0.753
S1	Centroid	0.778	-50	-11	0.770	0.752
NG8	Centroid	0.880	-75	-43	0.287	0.222
NG20	Ward	0.998	-75	-37	0.158	0.094

TABLE 4.2: Highest ARI, relative gain in memory and in time with sparsiciation

From this table we can see that all ARIs* are higher than those of SBC, implying a better clustering quality in SHCoClust. Moreover, ARIs* are obtained with significantly reduced memory usage and running time. Except for R5 dataset, in which median method only gains 8% in time and 25% in memory, other datasets obtain up to 86% time gain and 75% memory gain on average. This proves that SHCoClust is good at economizing computing resources, meanwhile preserving the clustering effectiveness. In other words, if given limited computing resources, SHCoClust is able to process relatively large datasets. This reflects the scalability of SHCoClust.

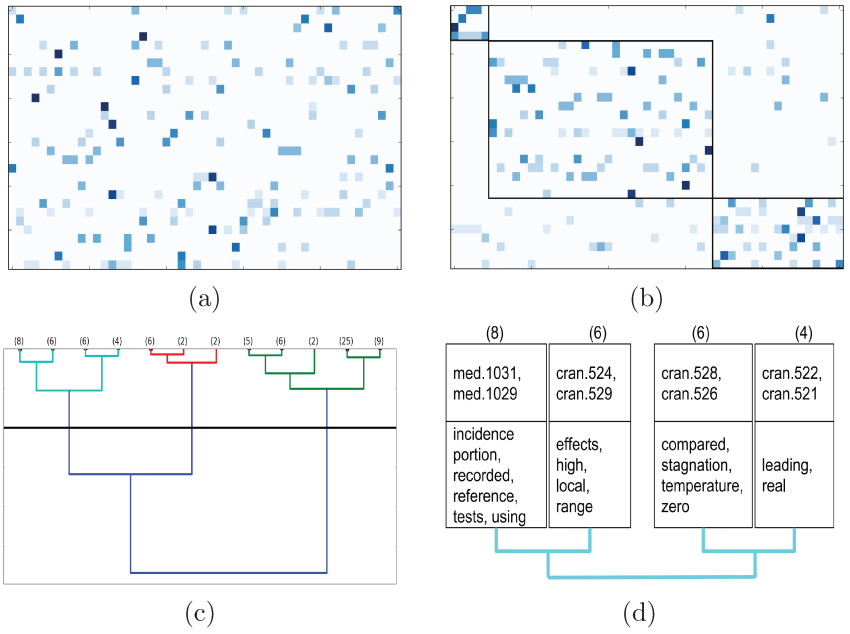


FIGURE 4.3: Doc-term matrix (a) before and (b) after SHCoClust, (c) dendrogram of co-clusters and (d) content the median-sized co-cluster

4.4 Visualization

We use a sampled dataset D_{sam} from the SMART collection to illustrate the hierarchies of co-clusters and the connections of sub-clusters inside a co-cluster. D_{sam} contains 30 documents (10 from each class) and 51 terms (after preprocessing). (a) and (b) of Figure 4.3 exhibit the doc-term matrix of D_{sam} before and after being processed by SHCoClust. In (b), we can clearly see that three co-clusters of different sizes lie along the diagonal. These co-clusters are resulted from the dendrogram cut at a height, shown in (c). Differing from dendrogram obtained by a conventional AHC method, dendrogram of SHCoClust has negative heights, thus it looks like it grows downwards. In (c), a leaf node of the dendrogram is marked by the number of members in it. The three sub-dendrograms above the cutting line (from left to right) map to the bottom-right, the top-left and the middle-situated co-clusters in (b). With the aid of (c), we are able to know that in (b) the bottom-right co-cluster is in fact closer to the top-left one and farther from the middle-situated co-cluster. (d) presents the content of the left-most sub-dendrogram in (c), i.e., the bottom-right co-cluster in (b). Keeping the same structure as it is in the dendrogram, (d) allows us to see in detail which documents and terms co-exit and how they are connected in their co-cluster.

4.5 Conclusion

In this chapter, a new algorithm, the similarity-based hierarchical co-clustering method, SHCoClust, is presented. We are interested in the advantageous property of co-clustering, which is capable to perform clustering in both data space and in feature space. Beginning with explaining the choice among several co-clustering techniques, we select the graph partitioning approach, as it allows to embed an arbitrary hard clustering method. The drawback of this kind of approaches is that they output flat co-clusters, and elements inside one co-cluster are not structured. Therefore, it is not possible to know how co-clusters are connected with each other, and how elements inside a co-cluster are organized. Inspired by the spectral embedding introduced in Section 2.3.3.3, we propose a hybrid learning method that is capable to perform hierarchical co-clustering. In such a way, co-clusters and elements inside an individual co-cluster are organized in a tree structure. It allows us to have a better understanding on an input dataset by examining how co-clusters and their elements are structured.

After presenting the motivation of this work, we present the proposed method by illustrating its computing procedure and the related mathematical details. We hope to give enough information to present a clear idea and method for readers. In the experiment part, we examine the proposed method from two aspects, the clustering effectiveness and computing efficiency. For the first aspect, we compare clustering quality of the proposed method with two benchmark methods in two scenarios, with a full-sized similarity matrix or with a sparsified one. By four different tests, it is concluded that the proposed method significantly improves clustering quality compared to the conventional AHC for most clustering methods, and it achieves improvements over SBC when sparsification is applied on the similarity matrix. As to computing efficiency, we examine the improvement of clustering efficiency when the similarity matrix is getting more and more sparsified, and how this influences clustering quality. Using linear and Gaussian kernels on three datasets, we discover that sparsifying similarity matrix can largely improve computing efficiency by reducing memory use and running time, more importantly, clustering quality does not compromise while computing efficiency is improved via sparsification. On the contrary, clustering effectiveness is improved when similarity matrix is sparsified.

Furthermore, we present a discussion on the complexity of the proposed method and a visualization on a sampled dataset. We show that the proposed method is capable to perform co-clustering, meanwhile, organizing co-clusters and elements inside individual co-cluster in a tree-like structure. It provides richer knowledge on the input data than either a pure co-clustering method or a pure hierarchical clustering method.

Mentioned in Section 4.3.4, for most cases in our experiments, the threshold value at percentile rank of 75% is the heuristically optimal threshold value, at which the ARI is preserved as high as possible while the similarity matrix is sparsified to have as less non-zero values as possible. A future work would be to determine a well-designed method to determine such a optimal threshold value for general cases when input datasets vary in size and in type.

Chapter 5

Testing the Cluster Hypothesis

5.1 Motivation

Previously, we present Sim_AHC and SHCoClust in Chapter 3 and in Chapter 4. As sated in Section 1.2 of Chapter 1, the task that our team undertakes is the “intelligent recursive and iterative information retrieval", for which we are required to apply clustering and co-clustering algorithms on IR tasks. However, IR is a wide field, it contains too many interesting research subjects that we do not have enough time for. Since our previous works are all clustering-based, we narrow down our research topic to the cluster hypothesis, which is the fundamental hypothesis of employing clustering methods in IR. By testing this hypothesis using Sim_AHC and SHCoClust, we are able to obtain important knowledge on how effectively a given query is responded, and with what efficiency. These knowledge, though theoretical, allow us to have basic but fundamental ideas on the retrieval behaviors of applying Sim_AHC and SHCoClust.

Since 1970s, many researchers have devoted their efforts to testing the cluster hypothesis from different aspects. Depending on the content of these works, we categorize these tests into classic tests, refined tests and language-model-based tests. For the classic tests, a group of research works emphasize comparisons of retrieval effectiveness between cluster-based and document-based retrieval systems; while some other works focus on testing the cluster hypothesis using hierarchical clustering methods; or on comparing cluster-based searching strategies in a hierarchy of clusters. The refined tests propose improved modification on some of the classic tests. Both classic tests and refined tests are based on the “bag-of-words" assumption, and they apply the vector space model to process their inputs. Differently, the language-model-based tests assume that documents are distributions of terms, thus the basic function that is used to measure similarities

among documents varies from the other tests. More details on these works can be found in Section 2.4 in Chapter 2.

Recall that the basic assumption adopted in our research for text clustering is the "bag-of-words" assumption and the vector space model. Therefore, the language-model-based tests are out of the scope of this thesis. As we devote much of our time to studying conventional AHC methods and the Lance-Williams formula, we are most interested in the cluster hypothesis tests that employ AHC methods. Interestingly, after carefully reviewing these tests, we find out that their conclusions are not consistent regarding which clustering method is the most effective. This inconsistency is likely caused by the differences in tested datasets, experimental settings and evaluation measures. In addition, these tests merely examine four AHC methods[1] in the Lance-Williams formula, leaving the retrieval performance of the other three methods unknown. Moreover, only retrieval effectiveness is addressed in these works, knowledge on retrieval efficiency is left blank. As AHC methods are computationally costly, it is practically important to examine their retrieval efficiency. These findings motivate us to design new tests that cover all AHC methods in the Lance-Williams formula within a universal experimental framework, and to examine the retrieval effectiveness and efficiency.

In the rest of this chapter, we elaborate on two proposed tests on the cluster hypothesis using Sim_AHC and SHCoClust, respectively. In each test, we employ optimal cluster search and the E-measure to evaluate retrieval effectiveness and compare it among seven AHC methods. Recall that a cluster-based retrieval process applies a search strategy and outputs a cluster that is believed to be the most relevant to a query. There are three search strategies, the top-down, the bottom-up and the optimal cluster search (Section 2.4.2.2). Among these strategies, we choose optimal cluster search in our experiments.

It is superior to its counterparts, as it does not depend on cluster representatives. Besides, it allows to compute a measure that balances between precision and recall, in order to avoid retrieving large clusters (that results in low precision but high recall) and small clusters (that have low recall but high precision). This measure used in optimal cluster search is called E-measure, it is defined as Equation 2.36. The characteristics of optimal cluster search and the E-measure allow us to evaluate retrieval effectiveness without bias, as they directly concern the structure of a dendrogram without depending on external object, such as a cluster representative.

In addition to examining and comparing retrieval effectiveness of different clustering methods, we also address efficiency issue. As Sim_AHC and SHCoClust are more efficient when their similarity matrices are sparsified, we test whether improving efficiency via sparsification influences the retrieval effectiveness measured by the E-measure. After

[1]These methods are: the single link, the complete link, the average link and the Ward method

presenting the two proposed tests, we illustrate an empirical comparison between them in order to have a better understanding on their retrieval performance.

The outline of this chapter is structured as follows: in Section 5.2, details on tested datasets, preprocessing and experiment setting are presented. In Section 5.3, a new test on the cluster hypothesis using Sim_AHC is introduced. Another test that uses SHCo-Clust is illustrated in Section 5.4. In each test, we compare the retrieval effectiveness of seven hierarchical clustering methods, and we examine the impact of improving efficiency via sparsification on the effectiveness. In Section 5.5, we compare between the two proposed tests by performing statistical tests, and discuss their computing complexity.

5.2 Datasets, Preprocessing and Experiment Setting

Five datasets from two collections are used in our experiments. Three of the Classic-4 datasets[2], MED, CISI and CACM, have been tested in previous works [93, 102, 112]. Introduction on MED (or MEDLINE) and CISI datasets can be found in Section 3.3.1 in Chapter 3. CACM dataset is a collection of abstracts from computer science journals. The other two datasets are sampled from Associated Press (AP) and Wall Street Journal (WSJ) collections, which are part of the TREC collection[3]. The full AP and WSJ collections in TREC contain 242,275 newswire articles and 161,512 journal articles, respectively. Table 5.1 lists the details of the five experimented datasets.

Collection	Classic-3			TREC	
Dataset	MED	CISI	CACM	AP	WSJ
nb.docs	1033	1460	2936	3147	1937
nb.terms	651	578	263	1562	1345
nb.queries	30	76	52	61	27
nb.docs/query	23	41	15	2	2

Table 5.1: Experimented datasets

For each dataset, a simple preprocessing is employed: stop words are removed; terms that have document frequency higher than 20% and lower than 2% are removed; the remaining terms are stemmed by Porter Stemmer [113]; TF-IDF weighting scheme is applied and l_2 normalization is performed on each document vector. The output of preprocessing for each dataset is a document-term matrix, whose number of documents and number of terms are indicated by *nb.docs* and *nb.terms* in Table 5.1.

[2]It contains the SMART datasets (MED, CISI and CRAN). We do not use CRAN (or CRANFIELD) dataset in the cluster hypothesis tests, because query indexes in its query file do not match those in its relevance judgment file. `http://ir.dcs.gla.ac.uk/resources/test_collections/`

[3]`https://catalog.ldc.upenn.edu/LDC93T3A`

Each dataset contains a complete query set and a relevance judgment file, which specifies a list of relevant documents to a query. In evaluation, we use the query set and the relevance judgment file to compute E-measure. The number of queries for each dataset is given by *nb.queries*, and the number of relevant documents per query is given by *nb.docs/query*.

For the TREC datasets (AP and WSJ), the relevance judgments files of TREC-2 are used[4]. They concern 100 queries indexed from 51 to 150 by the TREC convention. For AP collection, there are 97 queries extracted in total. On average, each query has 110 relevant documents. For WSJ collection, 50 queries are extracted, each query has 91 relevant documents on average. As our tested datasets are sampled, they contain less documents than the full collections, so we remove the documents that are not contained in these tested datasets from the list of relevant documents for each query. This results in 61 and 27 queries for the sampled AP and WSJ datasets, respectively. And each query has 2 relevant documents on average.

Our experiment is consisted of a number of tests. In each test, a full-sized or a sparsified pairwise similarity matrix S, generated by either linear kernel or Gaussian kernel, is used as input. We then apply one of the seven AHC methods (either in Sim_AHC or in SHCoClust) to produce a dendrogram, which is later cut at each height to output a set of flattened clusters. For each query in the query set, we search for its optimal cluster by scanning all the flattened clusters, for each of which we compute an E value. The cluster that has the minimal E value is the optimal cluster. The final output of such a test is a key-value pair, with a query ID being the key and the optimal E value (i.e., the E value of the optimal cluster that corresponds to the query) being the value.

Recall that E-measure [92] is used to evaluate retrieval effectiveness. Expressed as $E = 1 - \frac{(\beta^2+1)PR}{\beta^2 P+R}$, smaller E value indicates better retrieval effectiveness. P and R represent precision and recall, respectively, which can be expressed by $P = \frac{tp}{d}$ and $R = \frac{tp}{q}$. d is the number of documents in an optimal cluster, q is the number of relevant documents for a query, $tp = d \cap q$ is the number of true positive documents. β is the parameter that balances the importance between precision and recall, it takes values of 0.5, 1 and 2.

5.3 A New Cluster Hypothesis Test Using Sim_AHC

In this section, a new test on the cluster hypothesis is introduced. Differing from those tests that apply conventional AHC methods, this test applies the similarity-based hierarchical clustering framework, Sim_AHC. In Chapter 3, we mathematically and empirically

[4] `http://trec.nist.gov/data/qrels_eng/`

demonstrate that Sim_AHC is equivalent to the conventional AHC procedure with the Lance-Williams formula. Using inner product-based similarities, Sim_AHC allows to employ a threshold value to sparsify the similarity matrix in order to improve computing efficiency. Its experiments demonstrate that the sparsifying strategy not only improve efficiency, but also guarantee or improve clustering quality.

In this test, we apply Sim_AHC to testing the cluster hypothesis. The objective of this test is three-folded: first of all, we are interested in complementing previous research works, where only four AHC methods are tested. In this test, all conventional AHC methods that are unified in the Lance-Williams formula and in Sim_AHC are tested. Secondly, by applying optimal cluster search and computing the E-measure, we compare retrieval effectiveness among these methods and provide a benchmark using a universal experimental setting. Thirdly, we address efficiency issue in this test. Concretely, we examine the influence of sparsifying similarity matrix on retrieval efficiency, and on retrieval effectiveness.

5.3.1 Comparison of Retrieval Effectiveness Among Seven Clustering Methods

Table 5.3 and Table 5.4 display results of our first experiment. Each cell in Table 5.3 is an *averaged optimal E value*, which is the mean value of all optimal E values for a set of queries in a tested dataset. Suppose that a dendrogram is produced after applying the single link clustering method on MED dataset, whose query set contains 30 queries, $\mathbb{Q} = \{q_1, \ldots, q_{30}\}$. By optimal cluster search, we find an optimal cluster for a query and obtain the corresponding optimal E value. For this set of queries, the averaged optimal E value is computed as $\frac{1}{30}\sum_{i=1}^{30} E_i$, it is equal to 0.452 (at $\beta = 0.5$). Correspondingly, the standard deviation of optimal E values for the queries of MED dataset is 0.157, shown in Table 5.4.

In this experiment, the input similarity matrix S is generated by linear kernel and by Gaussian kernel. No thresholding strategy is applied, thus the full-sized S is used in computation. Values in Table 5.3 that are highlighted in bold are column-wise minimums. They signify the best retrieval effectiveness among seven clustering methods at a β value. As the averaged optimal E values are very close, we use three decimals to distinguish them. When these values are too close, we select the column-wise lowest using four decimals, but only three are shown. Likewise, in Table 5.4 column-wise minimal standard deviation is highlighted.

By observing Table 5.3, we can see that in both kernels Ward dominates the other methods with better retrieval effectiveness; McQuitty and average link achieve the best

effectiveness in some cases. A brief summary of Table 5.3 is provided in Table 5.2, which lists the clustering methods that obtain at least two lowest averaged E values at $\beta = 0.5, 1$ and 2.

Kernel/Dataset	MED	CISI	CACM	AP	WSJ
Linear kernel	Ward	Ward	Ward	Ward	McQuitty
Gaussian kernel	average	Ward	Ward	Ward	McQuitty

TABLE 5.2: Clustering methods that obtain at least two lowest averaged E values at $\beta = 0.5, 1$ and 2

In Table 5.4, we can see that standard deviation values are generally small, indicating that the overall difference between an optimal E value and the averaged optimal E value is small. Compared to MED, CISI and CACM datasets, AP and WSJ datasets have more larger standard deviation values. This is likely caused by the reduced number of relevant documents in their relevance judgment files. Small optimal clusters that contain relevant files tend to generate very low optimal E values, but larger optimal clusters that do not contain any relevant file tend to output very high optimal E values. This causes standard deviation values to increase in AP and WSJ datasets.

With Table 5.2, we can confirm that our results obtained from MED (using linear kernel), CISI, CACM and AP datasets conform to the conclusion in [100], which states that "Ward method was found to give the best overall results", though this work retrieves several clusters instead of one. As to the conclusions of [98, 102], which claim that "average link gave the best results", we only have the result obtained by MED dataset using Gaussian kernel to support them. In fact, compared to average link, we find that McQuitty achieves the same or better efficiency. Furthermore, our results disagree with the finding in [99], which concludes that "complete link is probably the most effective method". In our experiments, complete link performs poorly. Single link, centroid and median methods usually obtain larger averaged optimal E values than the other methods, showing poor retrieval effectiveness.

For two kernels, they obtain close results in many cases. Among all tested datasets, MED obtains generally lower E values than the other datasets, it is also the smallest collection. For MED, CISI and CACM datasets, their averaged optimal E values tend to increase as the values of β increase. On the contrary, for AP and WSJ datasets, as the values of β increase, averaged optimal E values tend to decrease. This is actually related to the sizes of optimal clusters and the number of relevant documents per query.

In MED, CISI and CACM datasets, the sizes of optimal clusters increase as the values of β increase. The size of an optimal cluster influences precision and recall. Larger optimal cluster tends to have lower precision but higher recall. When $\beta = 2$, E-measure assigns more importance to precision, making larger optimal clusters to have higher E values.

As to AP and WSJ datasets used in our experiments, the number of relevant documents per query is very small. In this case, larger clusters are likely to contain all relevant files, resulting in higher precision. Small clusters, on the other hand, probably have low precision as they do not contain any relevant document. This eventually makes E values to decrease when β moves from 0.5 to 2.

To compare among the averaged optimal E values obtained at $\beta = 0.5, 1$ and 2, we provide Figure 5.1 that illustrates the results of each dataset in Table 5.3. Numbers 1-7 along x-axis index seven clustering methods, from single link to Ward method. Dotted line with circle sign, solid line with triangle sign and dashed line with plus sign present the averaged optimal E values at $\beta = 0.5$, 1 and 2, respectively. Note that in order to clearly display the difference of the three lines, each plot has its own scale for y-axis.

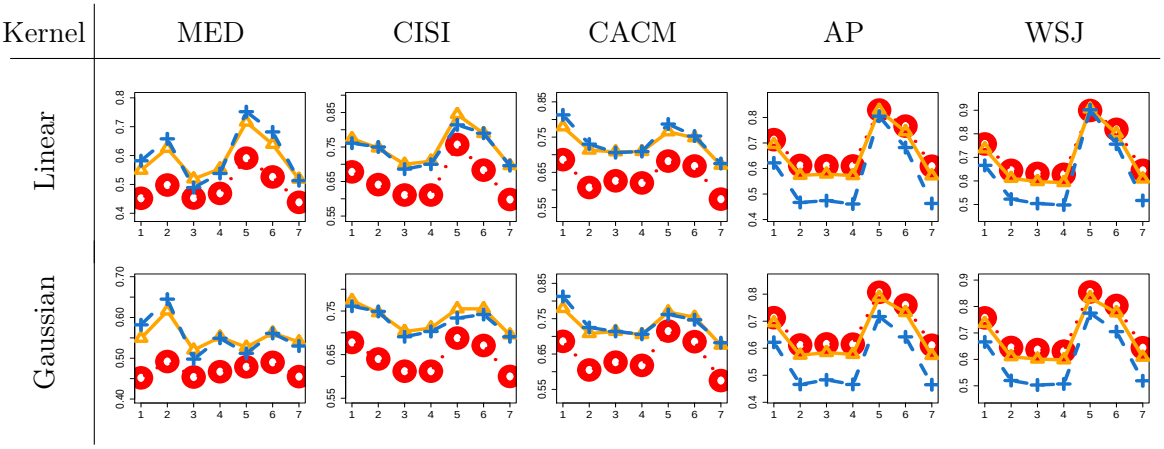


FIGURE 5.1: Illustration of results in Table 5.3 for each tested dataset

From Figure 5.1 we can see that, for MED, CISI and CACM datasets, the averaged optimal E values obtained at $\beta = 1$ and $\beta = 2$ are very close to each other, and they are higher than those obtained at $\beta = 0.5$. However, for AP and WSJ datasets, the averaged optimal E values obtained at $\beta = 0.5$ and $\beta = 1$ are close to each other, and farther from those obtained at $\beta = 2$. We can also observe that the three lines are fluctuating for all tested datasets. The lowest and the highest points along the three lines in each plot show the most effective and the least effective clustering methods for a dataset using a kernel function. There are a few common things shared by these plots: (1) the lowest point usually appears at integer 7, indicating that Ward method gives the best retrieval effectiveness in most cases; (2) the highest point often appears at integer 1 and 5, implying that single link and centroid method give the worst retrieval effectiveness; and (3) average link is as effective as McQuitty method in most cases, as the three lines are almost flat at integers 3 and 4.

	Dataset	MED			CISI			CACM			AP			WSJ		
Linear kernel	$\beta =$	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2
	single	0.452	0.549	0.582	0.678	0.774	0.761	0.686	0.779	0.813	0.713	0.694	0.622	0.757	0.736	0.666
	complete	0.498	0.625	0.658	0.641	0.747	0.750	0.607	0.713	0.730	0.612	0.573	0.467	0.645	0.611	0.523
	average	0.453	0.519	**0.489**	0.611	0.700	**0.686**	0.626	0.708	0.706	0.611	0.578	0.474	0.632	0.597	0.504
	McQuitty	0.469	0.554	0.538	0.611	0.709	0.700	0.620	0.709	0.709	0.610	0.572	**0.461**	**0.628**	**0.593**	**0.498**
	centroid	0.591	0.716	0.752	0.757	0.844	0.814	0.682	0.765	0.787	0.829	0.828	0.805	0.899	0.905	0.902
	median	0.526	0.639	0.682	0.683	0.789	0.790	0.668	0.748	0.753	0.766	0.744	0.682	0.819	0.801	0.756
	Ward	**0.438**	**0.518**	0.513	**0.598**	**0.696**	0.696	**0.574**	**0.670**	**0.675**	**0.609**	**0.571**	0.463	0.645	0.608	0.517

	Dataset	MED			CISI			CACM			AP			WSJ		
Gaussian kernel	$\beta =$	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2
	single	**0.452**	0.549	0.582	0.678	0.774	0.761	0.686	0.779	0.813	0.713	0.694	0.622	0.757	0.736	0.666
	complete	0.492	0.616	0.645	0.640	0.746	0.749	0.605	0.708	0.725	0.612	0.574	0.466	0.645	0.610	0.520
	average	0.454	**0.520**	**0.498**	0.612	0.703	**0.691**	0.627	0.714	0.714	0.616	0.584	0.483	0.637	0.601	**0.503**
	McQuitty	0.467	0.552	0.549	0.612	0.711	0.703	0.618	0.705	0.706	0.615	0.577	0.466	**0.631**	**0.598**	0.507
	centroid	0.479	0.527	0.512	0.688	0.755	0.734	0.715	0.769	0.762	0.807	0.788	0.717	0.855	0.833	0.775
	median	0.490	0.562	0.561	0.671	0.755	0.742	0.685	0.754	0.747	0.760	0.732	0.642	0.804	0.776	0.705
	Ward	0.455	0.540	0.530	**0.600**	**0.695**	0.691	**0.575**	**0.675**	**0.682**	**0.611**	**0.573**	**0.465**	0.644	0.608	0.519

TABLE 5.3: Retrieval effectiveness measured by averaged optimal E values for seven clustering methods in Sim_AHC using linear and Gaussian kernels

Linear kernel	Dataset	MED			CISI			CACM			AP			WSJ		
	$\beta =$	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2
	single	0.157	0.175	0.189	0.133	0.115	0.119	0.169	0.125	**0.116**	0.226	0.230	0.274	0.234	0.239	0.284
	complete	**0.144**	**0.153**	**0.155**	**0.120**	0.099	0.103	0.145	0.118	0.122	0.192	0.187	0.200	0.191	0.195	0.238
	average	0.163	0.179	0.198	0.136	0.118	0.120	0.136	0.112	0.126	0.207	0.199	0.212	0.174	**0.168**	0.206
	McQuttiy	0.153	0.167	0.175	0.125	0.109	0.109	**0.136**	**0.108**	0.125	0.197	0.192	0.211	0.175	0.169	**0.197**
	centroid	0.176	0.194	0.193	0.139	**0.091**	0.098	0.154	0.135	0.142	0.226	0.229	0.266	0.190	0.187	0.201
	median	0.166	0.177	0.181	0.144	0.102	**0.090**	0.148	0.110	0.120	0.216	0.223	0.272	0.189	0.206	0.260
	Ward	0.162	0.179	0.175	0.143	0.118	0.108	0.149	0.136	0.143	**0.187**	**0.179**	**0.191**	**0.167**	0.175	0.216

Gaussian kernel	Dataset	MED			CISI			CACM			AP			WSJ		
	$\beta =$	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2
	single	0.157	0.175	0.189	0.133	0.115	0.119	0.169	0.125	0.116	0.226	0.230	0.274	0.234	0.239	0.284
	complete	**0.144**	**0.159**	**0.163**	**0.121**	0.100	0.104	0.145	0.117	0.121	0.191	0.186	0.198	0.189	0.191	0.232
	average	0.164	0.179	0.190	0.132	0.117	0.119	0.139	0.116	0.127	0.204	0.196	0.214	0.171	**0.162**	**0.195**
	McQuitty	0.156	0.169	0.178	0.125	0.110	0.111	**0.135**	**0.107**	0.123	0.199	0.194	0.214	0.180	0.178	0.210
	centroid	0.178	0.206	0.212	0.140	0.108	0.101	0.145	0.127	0.137	**0.183**	0.181	0.218	**0.158**	0.168	0.196
	median	0.156	0.179	0.187	0.138	**0.098**	**0.090**	0.151	0.108	**0.110**	0.196	0.194	0.227	0.173	0.183	0.223
	Ward	0.158	0.182	0.181	0.145	0.117	0.110	0.150	0.141	0.147	0.188	**0.181**	**0.194**	0.169	0.178	0.218

TABLE 5.4: Standard deviation of optimal E values for seven clustering methods, corresponding to Table 5.3

5.3.2 Influence of Improving Efficiency via Sparsification on Retrieval Effectiveness

In Section 3.3.2 of Chapter 3, we observe that improving efficiency by sparsifying the similarity matrix S in Sim_AHC does not harm clustering quality, which on the contrary gets improved in many cases. We give two reasons to explain this phenomenon in Section 3.4: first of all, sparsifying S "purifies" similarities by removing non-significant values. Secondly, fusing a pair of similar clusters allows their respective neighborhoods to be combined, making it is more likely to absorb another cluster that is initially close to either of the neighborhoods.

Though measuring clustering quality is different from measuring retrieval effectiveness, these two associate with each other. Intuitively, better retrieval effectiveness is more likely obtained when documents are well clustered, in which documents in one cluster are highly similar and documents from different clusters are largely dissimilar. This motivates us to examine whether improving computing efficiency by sparsifying similarity matrix S influences retrieval effectiveness.

Like our previous experiments that involve sparsification, in this experiment, our baseline is still the absolute running time (in seconds) and memory usage when a full-sized similarity matrix S is used as input. We record the relative running time and relative memory usage when S is sparsified by a threshold value τ. Likewise, the set of τ values is selected from the $\{10, 25, 50, 75, 90\}\%$ percentile ranks of values in S. Results of this experiment are displayed in Figure 5.2 (when S is generated by linear kernel) and in Figure 5.3 (when S is generated by Gaussian kernel), where x-axis corresponds to the percentile ranks. Dotted line with circle sign and solid line with triangle sign represent the relative memory usage and the relative running time, respectively. Dashed lines with plus sign, cross sign and square sign indicate the averaged optimal E values at $\beta = 0.5, 1.0$ and 2.0, respectively.

When S is sparsified by a τ value, only its non-zero values are kept in memory. Performing a clustering method from Sim_AHC on a sparsified S outputs a reduced dendrogram, which is smaller than the dendrogram output by a full-sized S. This reduced dendrogram is cut at each height to generate a set of flat clusters, whose sizes vary from small to large. Flat clusters that are obtained near dendrogram's leaf nodes are usually smaller than those obtained near the dendrogram's root. For each query in the query set that is attached to the dataset in test, optimal cluster search is performed to find the optimal cluster for this query. Once the optimal cluster is found, an optimal E value is obtained, accordingly. In order to reflect how a specific dendrogram responds to all queries that

come from the same set, we average over their optimal E values to get the averaged optimal E value for the set of queries. This value is recorded and plot.

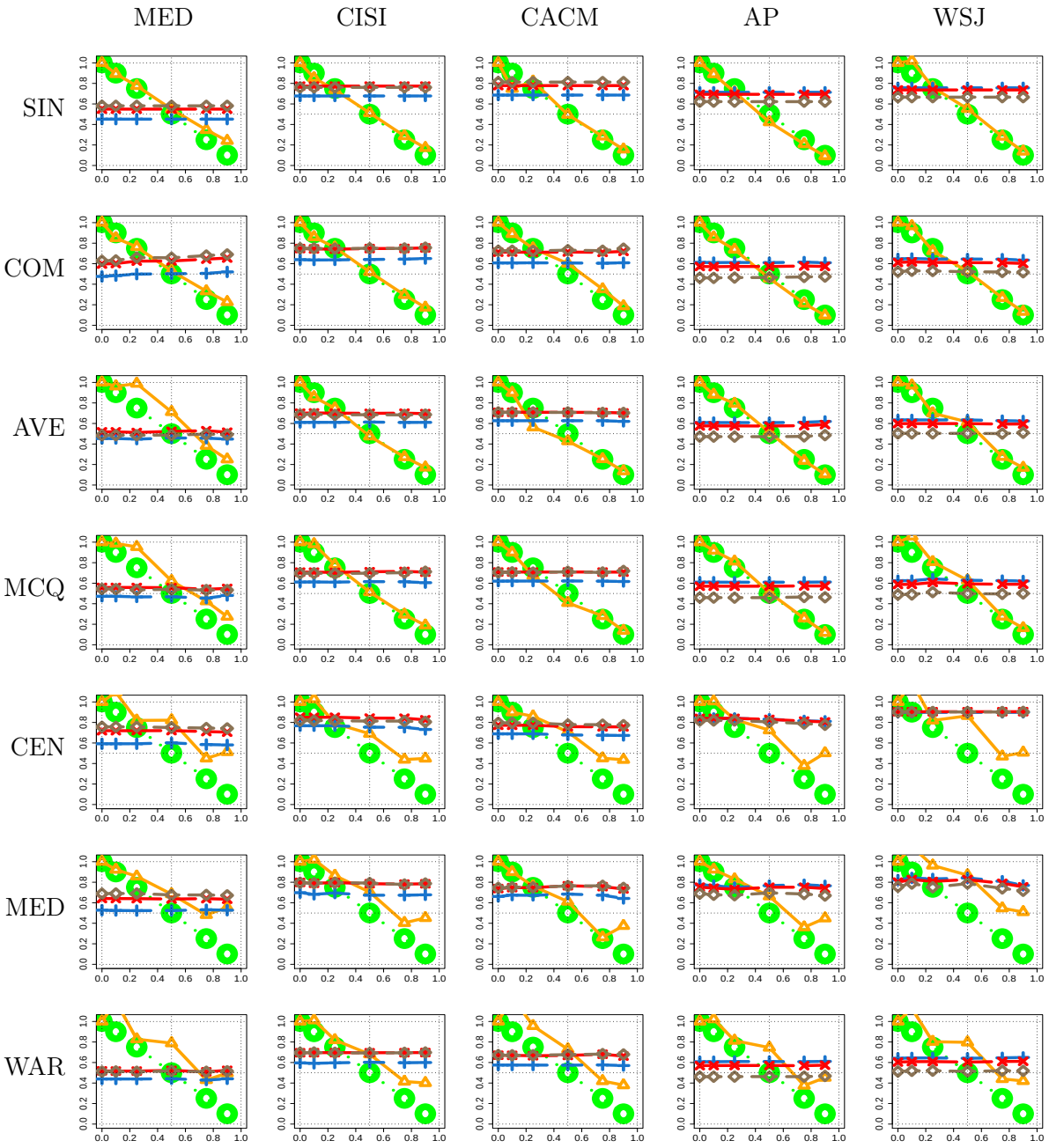


FIGURE 5.2: Results of sparsifying S obtained by linear kernel

From Figure 5.2 and Figure 5.3, we can see that retrieval effectiveness tends to be invariant to the effect of sparsification, because the lines of E values keep almost flat as relative memory use and running time gradually decrease when percentile ranks (along x-axis) increases. Some subtle fluctuations of E curves can be found in the plots of COM-MED and COM-CISI in Figure 5.2 and in Figure 5.3, in which three E curves lift up along the increase of percentile ranks. This indicates that retrieval effectiveness

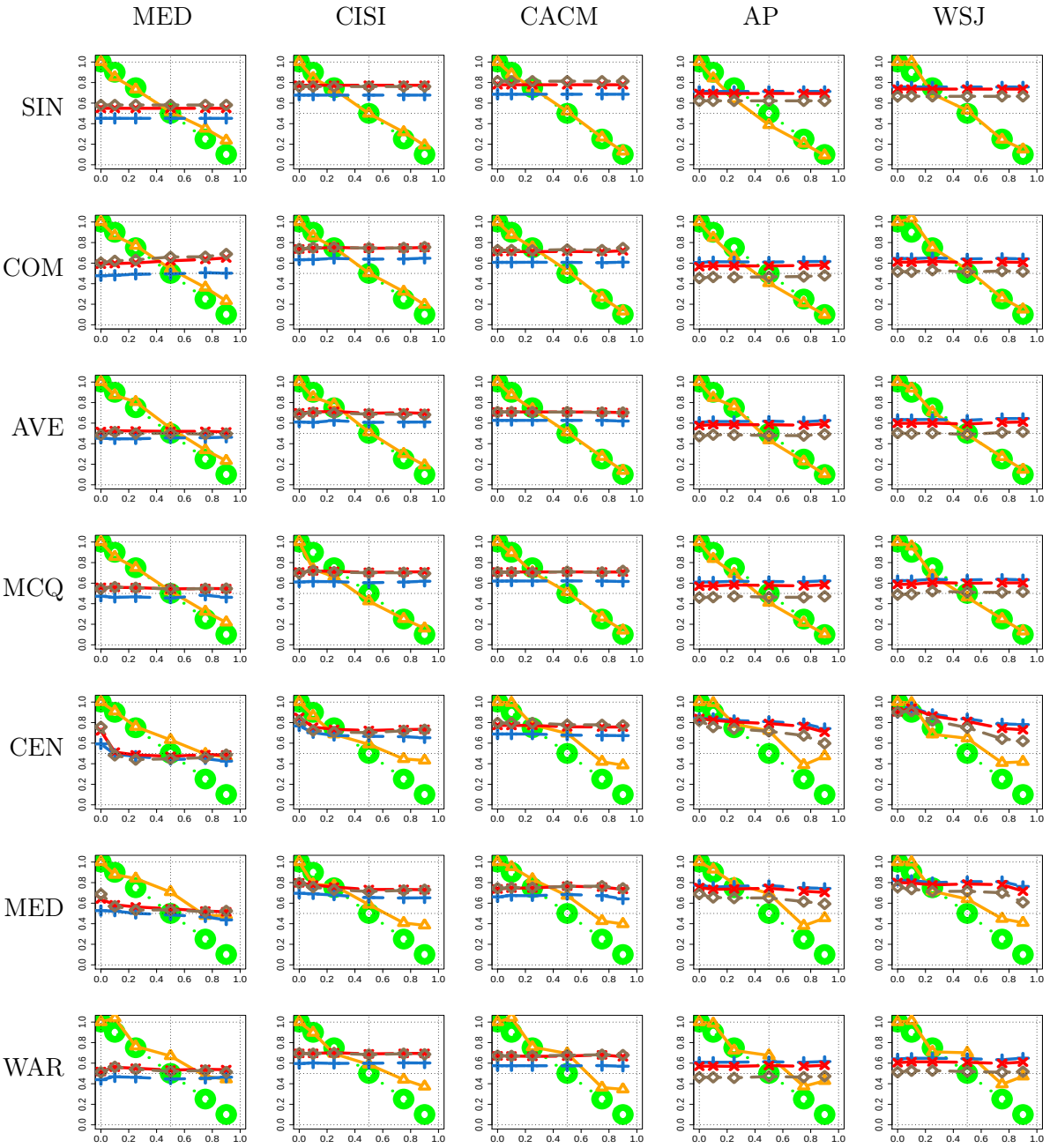


FIGURE 5.3: Results of sparsifying S obtained by Gaussian kernel

of using complete link in MED and CISI datasets becomes worse when the similarity matrix S is getting more and more sparsified. However, the opposite discovery can be found in the plots of centroid method applying on MED, CISI, AP and WSJ datasets in Figure 5.3, where E curves drop down when percentile rank is approaching to 1. It implies that even similarity matrix S is sparsified, with less memory use and less running time, retrieval effectiveness is actually improved.

In most plots, E curves tend to preserve at their levels and fluctuate very subtly as S

is being more and more sparsified. This is likely caused by the fact that most optimal clusters are small and they situate near the leaf nodes of a dendrogram. Sparsifying S mostly influences clusters that have small similarities, and these clusters are usually located near the root of the dendrogram. That is why sparsifying S does not really affect optimal clusters. And retrieval effectiveness is preserved. This is an interesting discovery. It implies that with sparsifying S, we can achieve the same retrieval effectiveness using much less computing resources.

Comparing two kernels, they obtain similar results. Though it is difficult to see much differences between Figure 5.2 and Figure 5.3, based on recorded values, we observe that Gaussian kernel always achieves better retrieval effectiveness than linear kernel for centroid and median methods. This holds for all datasets except for CACM dataset. In addition, both kernels obtain exactly the same results using single link for any full-sized or sparsified S.

5.3.3 Summary

In this section, we introduce a new test on the cluster hypothesis using the similarity-based hierarchical clustering framework, Sim_AHC. From the aspects of retrieval effectiveness and efficiency, we examine this test by comparing retrieval effectiveness among seven clustering methods, and by investigating the impact of improving efficiency via sparsification on retrieval effectiveness. In comparing seven clustering methods, we conclude that Ward outperforms the other methods. This conclusion agrees with some of past research works. In examining the influence of sparsifying similarity matrix on retrieval effectiveness, our experiments using linear kernel and Gaussian kernel demonstrate that improving computing efficiency by sparsifying similarity matrix does not harm retrieval effectiveness. In fact, retrieval effectiveness is almost invariant to the effect of sparsification. This result implies that using a largely sparsified similarity matrix, with substantially reduced memory and running time, retrieval effectiveness can be guaranteed. Comparing between linear kernel and Gaussian kernel, we find out that Gaussian kernel improves retrieval effectiveness of centroid and median methods.

5.4 A New Cluster Hypothesis Test Using SHCoClust

Unlike Sim_AHC, which performs clustering only on documents, SHCoClust groups both documents and terms at the same time. This characteristic makes it unique in the sense that it organizes co-clusters in a hierarchy and structures elements in a co-cluster. As stated previously, a cluster-based IR system built on such a method can be

advantageous, as it is capable to return relevant documents and "describing" terms of these documents. In this sense, richer knowledge is provided for seeking information. In previous section, by examining how well document clusters respond to a query in Sim_AHC, we obtain a few interesting findings. This makes us curious about the retrieval performance of SHCoClust. How well its document clusters respond to a query using a co-clustering method? Similarly, as the computing efficiency of SHCoClust can also be improved by sparsifying its similarity matrix, we are interested to see whether its retrieval effectiveness can be likewise guaranteed when sparsification is applied, as shown in the test of Sim_AHC. For such purposes, in this section, we perform a new test that applies SHCoClust to compare retrieval effectiveness among seven clustering methods, and to examine the impact of sparsifying similarity matrix on retrieval effectiveness.

Recall that in SHCoClust, a document-term matrix is firstly projected into a space that is constructed by eigenvectors of a graph Laplacian matrix, then clustering is performed on the projected data. Though the same set of datasets are used in this test, different processes are required by SHCoClust. First of all, spectral embedding is applied on a preprocessed document-term matrix to obtain matrix Z. Secondly, Z is normalized. Note that it is important to correctly map documents and terms to their corresponding vectors in Z. Thirdly, similarity matrix S_{co} is computed from Z and it is re-scaled to have similarity values between 0 and 1. The set of threshold values τ is selected from percentile ranks at $\{10, 25, 50, 75, 90\}\%$ of similarities in S_{co}. When a τ is used to sparsify S_{co}, a reduced dendrogram is produced. In order to correctly measure retrieval effectiveness based on relevance judgment information, it is necessary to remove all terms from an output dendrogram. Otherwise, the number of terms would increase the size of a cluster, lowering precision and affecting E value.

5.4.1 Comparison of Retrieval Effectiveness Among Seven Clustering Methods

In comparing retrieval effectiveness among seven clustering methods of SHCoClust, a full-sized similarity matrix S_{co} is used as input. With such an input, a clustering method produces a complete dendrogram, which is then cut at each height to generate a set of flat clusters. After removing terms from each flat cluster, optimal cluster search is performed on these flat clusters for each query to look for its optimal cluster and the optimal E value. The information provided by a relevance judgment file is used at this step to compute E values. The optimal E value is recorded for each query. However, in order to reflect how well a dendrogram responds to all queries of an input dataset, we average the optimal E values over the number of queries. This averaged optimal E value is used to evaluate the overall retrieval effectiveness of a clustering method on a dataset.

Table 5.6 and Table 5.7 illustrate the results of this experiment. A value in Table 5.6 is an averaged optimal E value, i.e., the mean value of optimal E values of all queries in a tested dataset. And a value in Table 5.7 is a corresponding standard deviation. Column-wise minimums in both tables are highlighted in bold. In Table 5.6, a bold value signifies the best retrieval effectiveness achieved by a clustering method at a β value. And in Table 5.7, a bold value marks the method that produces the most stable optimal E values at a β value.

To compare retrieval effectiveness among seven clustering methods from Table 5.6, a summary is provided in Table 5.5, in which the clustering method that achieves at least two minimums across three β values is listed for each dataset. When no such a method exists, a "$-$" sign is used. From this table, we can see that for linear kernel, average link outperforms the others in CISI and CACM datasets, with complete link for AP and WSJ datasets. For Gaussian kernel, the best performing method is Ward for CISI dataset, centroid for AP dataset, single link for CACM and WSJ datasets.

Kernel/Dataset	MED	CISI	CACM	AP	WSJ
Linear kernel	-	average	average	complete	complete
Gaussian kernel	-	Ward	single	centroid	single

TABLE 5.5: Clustering methods that obtain at least two lowest averaged optimal E values at $\beta = 0.5, 1$ and 2

It is interesting to see that the clustering methods that return the minimal averaged optimal E values in this test are very different from those in the test of Sim_AHC, where only Ward, McQuitty and average link are the winners. Results in this test provide a wider range of outperforming methods, especially for Gaussian kernel. However, one thing in common with the Sim_AHC test is that, as β increases from 0.5 to 2, the averaged optimal E values increase for MED, CISI and CACM datasets, and decrease for AP and WSJ datasets (Table 5.6). Likewise, this is influenced by the sizes of optimal clusters and the number of relevant documents per query.

By observing Table 5.3 and Table 5.6, we find that, in comparison with Sim_AHC, SHCoClust returns higher (averaged optimal) E values for MED, CISI and CACM datasets, and lower E values for AP and WSJ datasets. This implies that SHCoClust has better retrieval effectiveness than Sim_AHC for AP and WSJ datasets, but for the other datasets Sim_AHC has better retrieval effectiveness. And from Table 5.4 and Table 5.4, we can observe that, compared to Sim_AHC, SHCoClust usually obtains lower standard deviation for MED, CISI, CACM datasets and higher standard deviation for AP and WSJ datasets. There are two reasons to explain these: (1) in SHCoClust, the dendrogram of a clustering method grows by aggregating both documents and terms.

Compared to Sim_AHC, SHCoClust requires a larger number of iterations to complete clustering, and it eventually outputs a larger dendrogram, with documents and terms being its leaf nodes. When flattening this dendrogram to perform optimal cluster search, a larger number of flat clusters (with terms being removed) are generated, providing more candidates for the optimal cluster; (2) AP and WSJ datasets have only a few relevant documents per query. Therefore, it is more probable for a generated flat cluster to contain all relevant documents for a query. As SHCoClust generates more flat clusters than Sim_AHC, it is more likely to find small clusters that have precision equal to 1. Besides, as the number of relevant documents is small, recall tends to be high as well. MED, CISI and CACM datasets, on the other hand, have more relevant document per query. Therefore, it is like to have lower recall than AP and WSJ datasets.

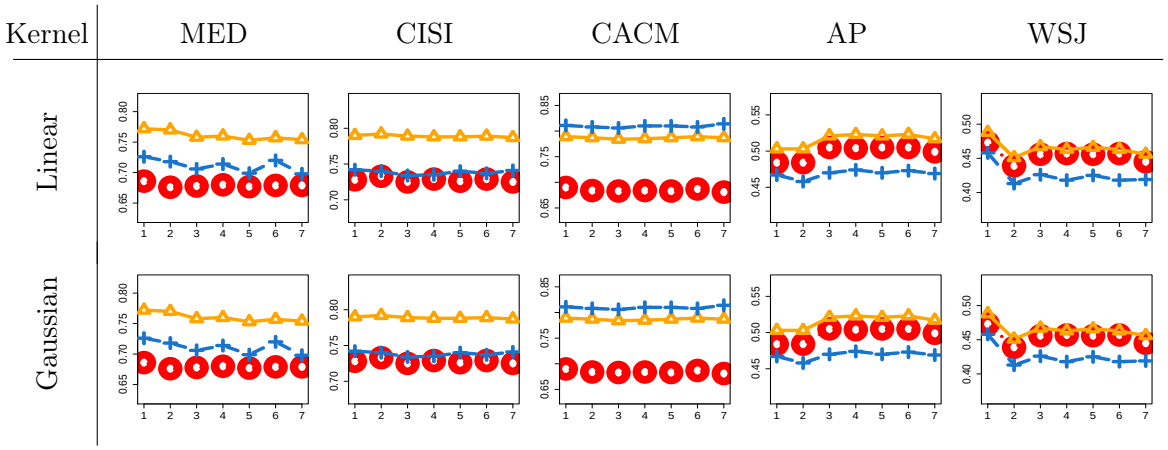


FIGURE 5.4: Illustration of results in Table 5.6 for each tested dataset

Figure 5.4 plots the results in Table 5.6. Like in Figure 5.1, integers 1-7 along x-axis map to seven clustering methods. Dotted line with circle sign, solid line with triangle sign and dashed line with plus sign represent the averaged optimal E values at $\beta = 0.5$, 1 and 2, respectively. Note that each plot has its own scale for y-axis. From Figure 5.4, we can observe that, all E curves are less fluctuate than in Figure 5.1 and tend to keep flat. This indicates that unlike in Sim_AHC, clustering methods in SHCoClust are close to each other in terms of retrieval effectiveness. One thing in common to Figure 5.1 is that, E values at $\beta = 0.5$ are lower than those at $\beta = 1$ and 2 in MED, CISI and CACM datasets. In AP and WSJ datasets, however, E values at $\beta = 2$ is lower than those at $\beta = 0.5$ and 1. In particular, in MED dataset, E curves are better separated in SHCoClust than in Sim_AHC.

Linear kernel	Dataset	MED			CISI			CACM			AP			WSJ		
	$\beta =$	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2
	single	0.686	0.772	0.726	0.728	0.790	0.742	0.690	0.789	0.811	**0.484**	0.503	0.467	0.473	0.488	0.458
	complete	**0.676**	0.770	0.718	0.733	0.792	0.740	0.684	0.787	0.808	0.484	**0.503**	**0.458**	**0.439**	**0.451**	**0.413**
	average	0.678	0.758	0.706	**0.725**	0.789	**0.734**	0.683	**0.784**	**0.806**	0.505	0.521	0.470	0.456	0.467	0.426
	McQuitty	0.680	0.760	0.714	0.729	0.788	0.735	0.684	0.785	0.810	0.504	0.523	0.474	0.457	0.463	0.418
	centroid	0.677	**0.753**	0.699	0.726	0.788	0.740	0.683	0.787	0.810	0.505	0.521	0.470	0.456	0.466	0.425
	median	0.679	0.757	0.720	0.729	0.789	0.737	0.687	0.789	0.808	0.505	0.523	0.473	0.457	0.463	0.418
	ward	0.679	0.754	**0.698**	0.725	**0.787**	0.741	**0.681**	0.787	0.814	0.499	0.517	0.469	0.445	0.456	0.419

Gaussian kernel	Dataset	MED			CISI			CACM			AP			WSJ		
	$\beta =$	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2
	single	0.673	0.759	0.749	0.733	0.818	0.800	**0.675**	**0.774**	0.797	0.525	0.562	0.529	**0.493**	**0.502**	**0.453**
	complete	0.645	0.744	0.707	0.722	0.798	0.767	0.677	0.778	0.796	0.514	0.539	0.490	0.566	0.559	0.498
	average	0.645	0.735	**0.679**	0.722	0.788	0.745	0.683	0.782	0.796	0.525	0.546	0.491	0.566	0.559	0.498
	McQuitty	0.651	**0.730**	0.687	**0.721**	0.788	0.747	0.680	0.778	0.793	0.525	0.548	0.499	0.556	0.546	0.469
	centroid	**0.643**	0.734	0.687	0.723	0.790	0.752	0.687	0.781	0.796	**0.506**	**0.529**	0.483	0.568	0.560	0.496
	median	0.643	0.737	0.703	0.722	0.790	0.750	0.682	0.778	**0.789**	0.512	0.532	**0.480**	0.552	0.545	0.477
	Ward	0.650	0.740	0.691	0.728	**0.787**	**0.744**	0.678	0.780	0.795	0.514	0.536	0.485	0.539	0.537	0.480

TABLE 5.6: Retrieval effectiveness measured by averaged optimal E values for seven clustering methods in SHCoClust using linear and Gaussian kernels

Linear kernel	Dataset	MED			CISI			CACM			AP			WSJ		
	$\beta =$	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2
	single	0.079	0.066	0.088	0.119	0.119	0.133	0.139	**0.119**	**0.119**	0.249	0.247	0.278	0.318	0.303	0.304
	complete	0.075	**0.057**	0.090	**0.090**	**0.090**	**0.121**	0.142	0.132	0.132	0.232	0.222	0.256	0.309	0.292	0.291
	average	0.076	0.064	0.092	0.118	0.117	0.133	0.139	0.130	0.130	0.232	0.219	0.255	0.299	0.283	0.285
	McQuitty	0.075	0.064	**0.085**	0.118	0.116	0.131	0.143	0.131	0.132	0.233	0.219	0.251	0.298	0.280	0.281
	centroid	0.076	0.072	0.099	0.118	0.117	0.128	0.139	0.122	0.120	0.233	0.220	0.255	0.298	0.282	0.283
	median	0.075	0.067	0.086	0.119	0.117	0.131	0.138	0.121	0.120	0.233	0.218	**0.250**	**0.298**	**0.280**	**0.281**
	Ward	**0.073**	0.072	0.092	0.118	0.116	0.130	**0.137**	0.121	0.123	**0.228**	**0.216**	0.251	0.315	0.297	0.291

Gaussian kernel	Dataset	MED			CISI			CACM			AP			WSJ		
	$\beta =$	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2
	single	0.100	0.111	0.106	0.131	0.130	0.141	**0.126**	0.138	0.169	0.259	0.227	0.251	0.304	0.284	0.288
	complete	0.095	0.092	**0.101**	**0.097**	**0.097**	**0.121**	0.134	0.137	0.155	0.212	0.187	0.232	0.282	0.262	0.267
	average	0.093	0.096	0.121	0.098	0.098	0.123	0.134	0.137	0.155	0.213	0.185	**0.223**	0.276	0.263	0.281
	McQuitty	0.097	0.093	0.102	0.100	0.102	0.128	0.139	0.139	**0.154**	0.233	0.206	0.235	0.272	**0.242**	**0.245**
	centroid	0.105	0.109	0.117	0.099	0.098	0.122	0.134	**0.137**	0.154	0.231	0.209	0.245	**0.270**	0.254	0.273
	median	0.098	0.105	0.118	0.099	0.100	0.128	0.140	0.140	0.155	0.235	0.210	0.241	0.275	0.250	0.263
	Ward	**0.090**	**0.086**	0.104	0.101	0.100	0.127	0.136	0.139	0.155	**0.209**	**0.184**	0.232	0.286	0.270	0.277

TABLE 5.7: Standard deviation of optimal E values for seven clustering methods, corresponding to Table 5.6

5.4.2 Impact of Sparsification on Retrieval Effectiveness

In the test of Sim_AHC, shown in Section 5.3.2, we find out that retrieval effectiveness is guaranteed when similarity matrix is getting more and more sparsified. Unlike sparsifying a similarity matrix in Sim_AHC, in SHCoClust sparsification is applied on a similarity matrix that is obtained from projected data. This difference might lead to different conclusions from the experiment of Sim_AHC. Interested in finding this out, in this section we examine the influence of sparsification on retrieval effectiveness in SHCoClust.

Likewise, the baseline of our experiment is the absolute running time $T_{\tau=0}$ and memory use $M_{\tau=0}$ when a full-sized S_{co} is taken as input. Given a threshold value τ, $\tau > 0$, S_{co} is sparsified by only keeping its non-zero values that are greater or equal to τ in memory. A clustering method is applied on a sparsified S_{co} and an incomplete dendrogram is output. The memory use that is occupied by the sparsified S_{co}, M_τ, and the running time of performing clustering, T_τ, are recorded. The relative memory use, $\frac{M_\tau}{M_{\tau=0}}$, and the relative running time, $\frac{T_\tau}{T_{\tau=0}}$, are computed and plotted. The output dendrogram is flattened at each height, resulting a set of flat clusters. In order to compute E-measure, only documents are kept in each flat cluster (terms are removed). Using the relevance judgment information in a dataset, the optimal cluster is searched for each query and corresponding optimal E value is computed. In the end, the averaged optimal E value is calculated for all queries.

Figure 5.5 and Figure 5.6 illustrate experiment results of sparsifying S_{co}, which is obtained using linear kernel and Gaussian kernel, respectively. Like in Figure 5.2 and in Figure 5.3, percentile ranks are along x-axis; (green) dotted line with circle sign represents relative memory use, $\frac{M_\tau}{M_{\tau=0}}$, and (orange) solid line with triangle sign denotes relative running time, $\frac{T_\tau}{T_{\tau=0}}$. Dashed lines with plus sign, cross sign and square sign indicate the averaged optimal E values at $\beta = 0.5$, 1 and 2, respectively.

It is interesting to see that a similar conclusion can be drawn from this experiment as in the test of Sim_AHC, that is, retrieval effectiveness tends to be invariant to the effect of sparsifying similarity matrix. It is clear that, in the results of both linear and Gaussian kernels, the lines of the averaged optimal E values are almost straight in all plots, in which relative running time and relative memory use decrease as percentile rank increases. In fact, these lines are even more constant than the results obtained in the same experiment of Sim_AHC. In addition, results obtained from two kernels are also very close.

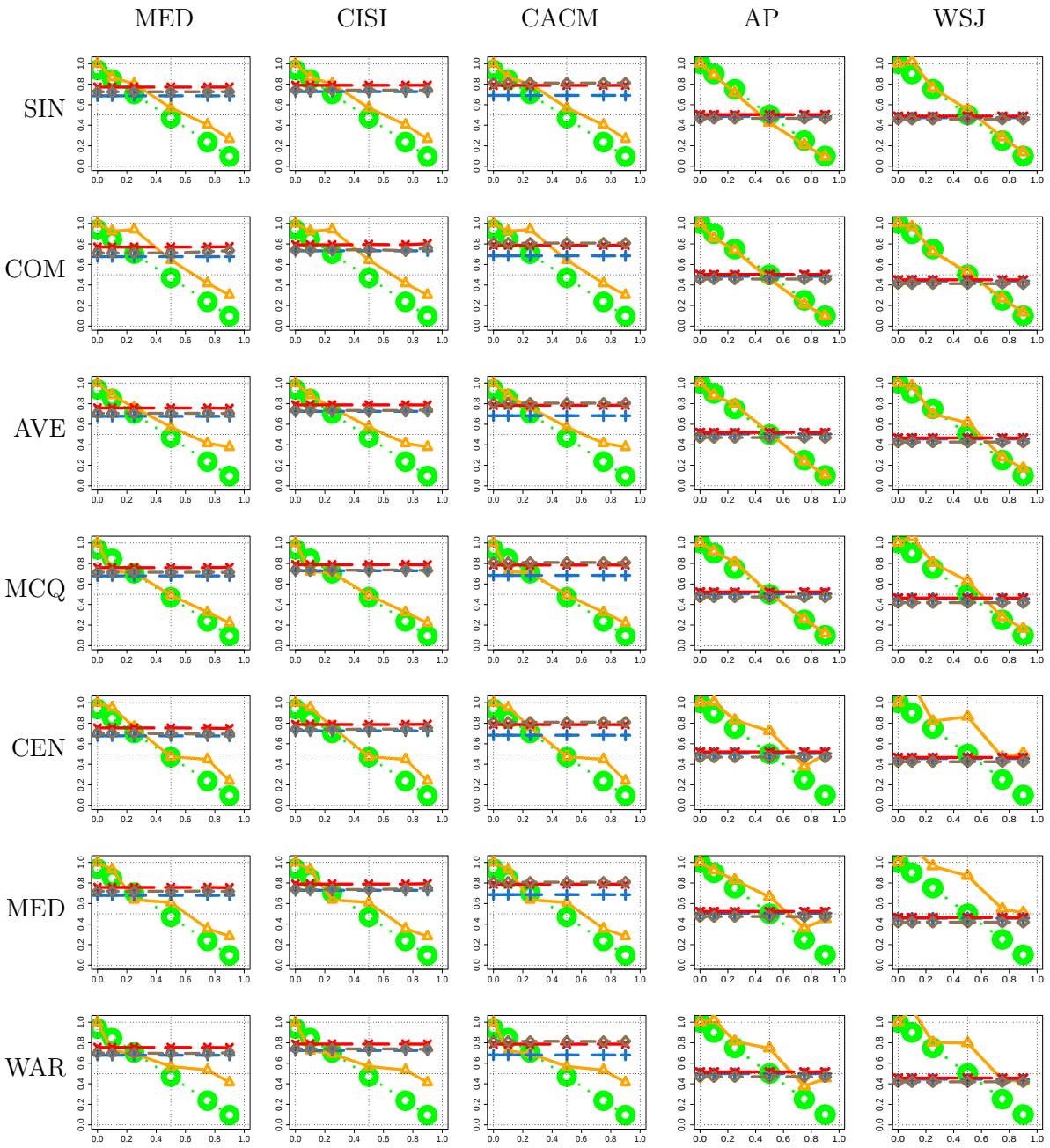


FIGURE 5.5: Results of sparsifying S_{co} obtained by linear kernel

5.4.3 Summary

In this section, we propose a new test on the cluster hypothesis using SHCoClust. In our experiments, we firstly illustrate results of comparing retrieval effectiveness among seven clustering methods, then we examine the impact of sparsifying similarity matrix on retrieval effectiveness and on computing efficiency. In the first experiment, we find out that Ward method is no more the dominating one as concluded in the test of Sim_AHC.

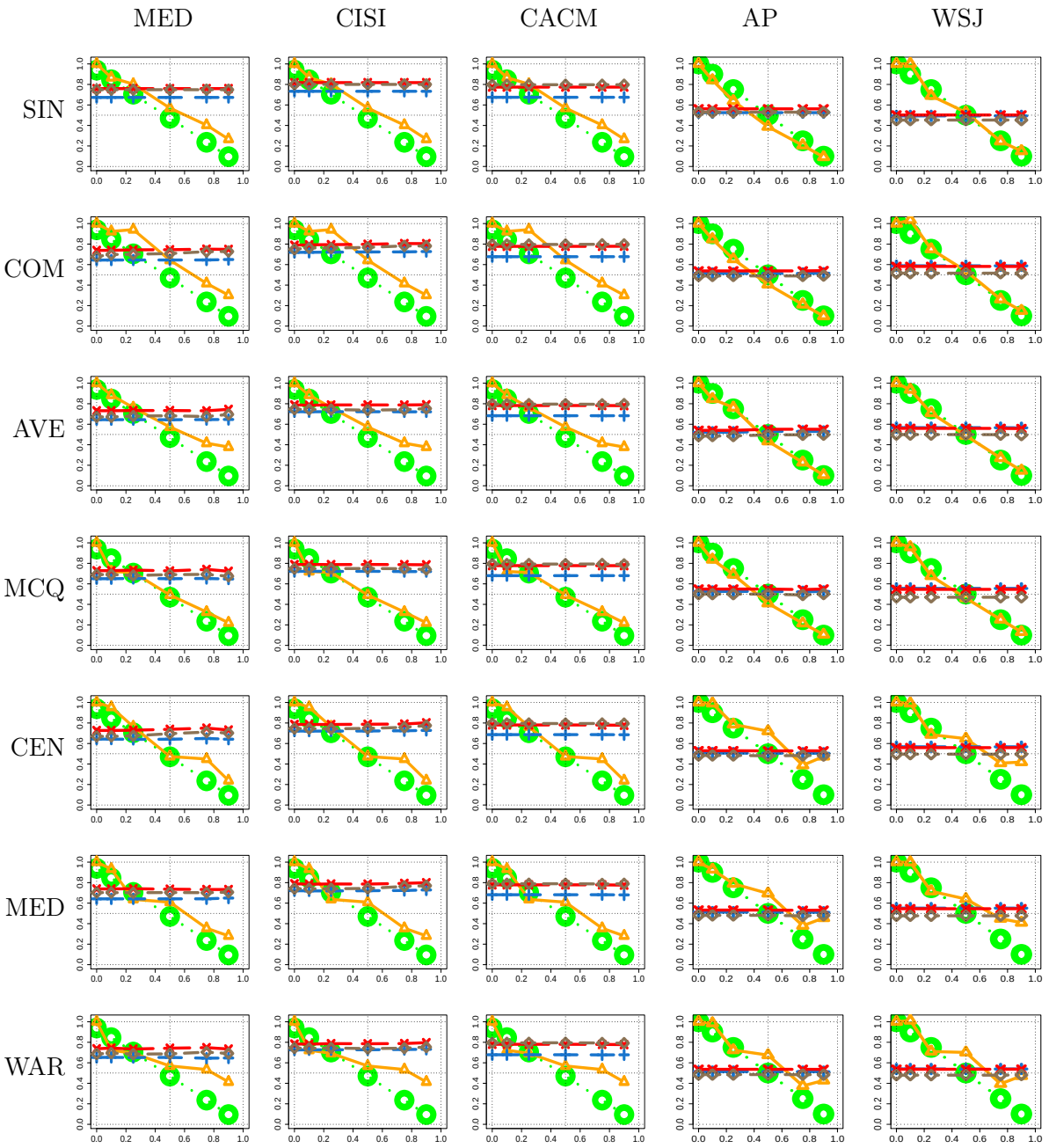


FIGURE 5.6: Results of sparsifying S_{co} obtained by Gaussian kernel

A wider range of methods are shown to be performing. However, the difference of the averaged optimal E values obtained by seven methods is in fact quite small. In the second experiment, we obtain a similar conclusion, i.e., sparsifying similarity matrix results in better computing efficiency without harming retrieval effectiveness. This is probably caused by the fact that most optimal clusters are located near the leaf nodes of an output dendrogram. Sparsification disconnects clusters that have small similarities, and these clusters are likely near the root. Therefore, optimal clusters are not affected by sparsifying the similarity matrix, and retrieval effectiveness keeps invariant as threshold

value increases. Unlike Sim_AHC, SHCoClust goes through more iterations and outputs larger dendrograms. This results in more flat clusters for optimal cluster search, and provides more candidates to be selected as the optimal cluster for a query. It is why SHCoClust has lower variance of the optimal E values than Sim_AHC.

5.5 Comparison between Two Proposed Tests

In Section 5.3 and Section 5.4, we design and carry out two new tests on the cluster hypothesis using Sim_AHC and SHCoClust, respectively. In each test, we compare retrieval effectiveness among seven clustering methods and examine the impact of improving computing efficiency by sparsification on retrieval effectiveness. In this section, we compare the two tests to find out which clustering framework has better retrieval effectiveness. Besides, we provide a discussion on the complexity of the two clustering frameworks.

5.5.1 On Retrieval Effectiveness

One way to compare retrieval effectiveness between Sim_AHC and SHCoClust is to compare results in Table 5.3 and in Table 5.6. In order to perform a fair comparison, we carry out a set of paired two-tailed Student T-tests between the results obtained in the Sim_AHC test and the results obtained in the SHCoClust test. Our objective is to examine whether the retrieval effectiveness of Sim_AHC is the same or significantly different from that of SHCoClust. If they are significantly different, we compare their averaged optimal E values to conclude which one is more effective.

Previously, it is explained that the averaged optimal E value is the mean value of a list of optimal E values for a set of queries. To perform one T-test, we take two lists of optimal E values. One list is from Sim_AHC and the other list is from SHCoClust using the same clustering method on the same dataset with the same kernel function at the same β value. The null hypothesis is $H_0 : \mu_0 = \mu_1$, with μ_0 and μ_1 denoting the averaged optimal E value of Sim_AHC and of SHCoClust, respectively. And the alternative hypothesis is $H_1 : \mu_0 \neq \mu_1$. It is a paired T-test, because the two lists of optimal E values in comparison are generated from the same dataset for the same query set using the same clustering method, but from two different frameworks.

In each T-test, we compute a statistical T value and compare it against the corresponding critical value at confidence level $\alpha = 95\%$. If the absolute T value is greater than its critical value, the null hypothesis H_0 is rejected and the alternative hypothesis H_1 is

accepted; otherwise, the alternative hypothesis H_1 is rejected and the null hypothesis H_0 is accepted. Table 5.8 displays T values of all T-tests, with those whose absolute value are smaller than critical value being highlighted in red.

From Table 5.8, we can observe that most (absolute) T values are greater than critical values, implying that in most cases the retrieval effectiveness of Sim_AHC is significantly different from that of SHCoClust. In some cases, (absolute) T values are smaller than critical values. As we can see, for MED, CISI and CACM datasets, red T values often appear at single link, median and centroid methods. However, for AP and WSJ (using Gaussian) datasets, red T values occur in complete link, average link, McQuitty and Ward methods.

For T-tests that accept H_1 in MED, CISI and CACM datasets, we further compare the averaged optimal E values of Sim_AHC (Table 5.3) and SHCoClust (Table 5.6), we conclude that for these datasets Sim_AHC is significantly more effective than SHCoClust using complete link, average link, McQuitty and Ward methods. On the other hand, for T-tests that accept H_1 in AP and WSJ datasets, we find that SHCoClust is significantly more effective than Sim_AHC using single link, centroid and median methods.

	Critical value	2.045			1.992			2.009			2.000			2.056		
Linear kernel	Dataset	MED			CISI			CACM			AP			WSJ		
	$\beta =$	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2
	single	-7.901	-7.270	-5.038	-3.231	-1.150	1.660	-0.138	-0.426	0.060	5.173	4.116	3.007	4.311	3.905	3.232
	complete	-7.191	-5.698	-2.271	-6.804	-4.261	0.603	-3.810	-5.094	-5.506	3.265	1.885	0.257	3.420	2.834	2.180
	average	-6.736	-7.234	-6.786	-6.822	-6.458	-4.082	-2.533	-4.253	-5.911	2.630	1.479	0.122	3.334	2.842	2.237
	McQuitty	-7.146	-6.566	-5.780	-7.566	-5.897	-2.792	-2.582	-3.930	-5.599	2.624	1.249	-0.324	2.815	2.368	1.766
	centroid	-2.596	-1.018	1.495	1.428	3.058	4.054	-0.031	-1.073	-1.285	8.559	7.472	6.809	5.669	5.653	5.863
	median	-5.127	-3.880	-1.309	-2.388	-0.045	3.128	-0.763	-2.298	-4.124	6.760	5.252	4.231	5.594	5.152	4.490
	Ward	-7.512	-7.603	-7.467	-6.676	-5.407	-3.089	-4.772	-5.769	-6.872	2.924	1.485	-0.175	3.388	2.788	2.059

	Dataset	MED			CISI			CACM			AP			WSJ		
Gaussian kernel	$\beta =$	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2	0.5	1	2
	single	-6.430	-5.959	-4.481	-2.960	-2.858	-3.200	0.393	0.163	0.529	3.809	2.580	1.609	3.238	2.909	2.582
	complete	-5.295	-4.667	-2.339	-5.312	-4.050	-1.701	-3.394	-4.257	-5.248	2.776	1.057	-0.744	0.891	0.459	0.069
	average	-6.270	-6.839	-6.010	-6.747	-6.288	-4.762	-2.581	-3.603	-4.884	2.423	1.075	-0.240	1.154	0.709	0.094
	McQuitty	-6.113	-6.135	-4.851	-7.062	-5.776	-3.705	-2.663	-3.884	-5.528	2.238	0.748	-0.881	1.317	0.999	0.847
	centroid	-4.618	-5.934	-5.643	-1.743	-2.142	-1.221	1.067	-0.410	-1.186	9.189	7.682	5.918	5.281	4.953	4.525
	median	-5.470	-5.998	-5.180	-2.746	-2.258	-0.520	0.123	-1.054	-1.950	7.005	5.532	4.020	4.143	3.728	3.269
	Ward	-6.708	-6.704	-6.394	-6.920	-5.970	-4.240	-4.469	-5.844	-7.218	2.792	1.123	-0.628	1.741	1.241	0.752

TABLE 5.8: T values of T-tests with $H_0 : \mu_0 = \mu_1$ and $H_1 : \mu_0 \neq \mu_1$ at $\alpha = 95\%$. μ_0 indicates the mean of optimal E values obtained in Sim_AHC, μ_1 for SHCoClust. Values highlighted in red are smaller than critical value.

5.5.2 On Computing Efficiency

In Section 3.3.2.2 and Section 4.3.4, we provide some discussion on the time complexity of Sim_AHC and of SHCoClust. Given a document-matrix A of shape $n \times m$, without sparsification, the time complexity of Sim_AHC is $O(n^3)$, and $O(\min(nm^2, mn^2) + (n+m)^3)$ for SHCoClust. As shown previously, better computing efficiency can be achieved by sparsifying similarity matrix S, resulting in M non-zero values being stored in memory, $M \ll n^2$. This strategy reduces time complexity of Sim_AHC to $O(nM)$, and of SHCoClust to $O(\min(nm^2, mn^2) + (n+m)M)$. Comparing the two clustering frameworks, SHCoClust is more costly than Sim_AHC no matter sparsifying S or not, because it requires to apply spectral embedding and to perform clustering on a similarity matrix of shape $(n+m) \times (n+m)$. However, it allows to retrieve both documents and terms, which Sim_AHC cannot do.

5.5.3 Summary

In this section, we compare retrieval effectiveness between Sim_AHC and SHCoClust by performing a set of paired two-tailed T-tests. We conclude that Sim_AHC is more effective than SHCoClust using complete link, average link, McQuitty and Ward methods in MED, CISI and CACM datasets. On the contrary, SHCoClust is more effective than Sim_AHC using single link, centroid and median methods in AP and WSJ datasets. In terms of computing efficiency, determined by the computing procedures, SHCoClust is more costly than Sim_AHC.

5.6 Conclusion

In this chapter, two new tests on the cluster hypothesis are performed using Sim_AHC and SHCoClust. The motivation of performing these tests is to obtain important knowledge on how effectively a query is responded using different clustering methods, and with what computing efficiency. We believe that this knowledge is fundamental for us to understand the retrieval behaviors of the proposed clustering frameworks. Another interest of this work is to provide a benchmark on this topic. In reviewing past works, we find out that conclusions drawn from the cluster hypothesis tests are not consistent, in comparing retrieval effectiveness among several hierarchical clustering methods. The inconsistency is likely caused by the difference in datasets, experiment setting and evaluation. Additionally, in the past works, only four out of seven conventional clustering methods are tested. This motivates us to propose new tests, in which we use universal datasets, experiment settings and evaluation measure to test seven clustering

methods in the framework of Sim_AHC and of SHCoClust. As Sim_AHC is equivalent to conventional AHC and the Lance-Williams formula, retrieval effectiveness obtained by Sim_AHC is the same to the conventional AHC framework. Another contribution of this work is that we provide insight of retrieval efficiency for the cluster hypothesis tests that apply hierarchical clustering methods. Concretely, we examine the impact of improving efficiency by sparsifying similarity matrix on retrieval effectiveness.

In the test that uses Sim_AHC, we discover that Ward method is the dominating method, who achieves the best retrieval effectiveness in most cases. This conclusion agrees with some of the past works. However, in the test that uses SHCoClust, a wider range of performing methods is returned. As to the impact of improving efficiency by sparsifying similarity matrix on retrieval effectiveness, both tests present similar result, i.e., retrieval effectiveness is almost invariant to the impact of sparsification. We observe that when threshold value approaches to 1, with reduced memory use and running time, retrieval effectiveness is guaranteed. This is an interesting discovery, which implies that it is likely to achieve the same retrieval effectiveness with largely reduced memory use and running time. We explain that sparsification disconnects clusters that are near the root of a dendrogram, but it gives little impact on most optimal clusters, who are close to the leaf nodes of the dendrogram. In comparing the retrieval effectiveness between the proposed tests, we perform a sets of paired two-tailed T-tests. We conclude that Sim_AHC is significantly more effective than SHCoClust using complete link, average link, McQuitty and Ward methods. However, SHCoClust is more effective than Sim_AHC using single link, centroid and median methods.

Chapter 6

The Distributed Implementations

6.1 Introduction

Previously, we demonstrate that the computing efficiency of Sim_AHC and SHCoClust can be improved by sparsifying their similarity matrix. This strategy allows Sim_AHC and SHCoClust to address relatively larger datasets with limited computing resources on a single machine, where calculation is processed sequentially. Introduced in Section 2.1.2, distributed and parallel computing are capable to take advantage of hardware to improve efficiency by assigning computing tasks among a set of processors. These processors can be either located in one machine (parallel), or connected via a local or remote network (distributed). For each type of computing, there exists a number of softwares.

We improve the efficiency of Sim_AHC and SHCoClust from an algorithmic point of view. Interested in further enhancing their efficiency, we choose distributed computing to accelerate the speed of calculation in case of processing larger input. Compared parallel computing, distributed computing is capable to handle issues such as data replication, machine failure, job scheduling and recovery. These functions save us time from many extra problems in implementing our programs. This is the main reason that we choose distributed computing.

Among many distributed computing softwares, we choose Apache Spark. Mentioned in Section 2.1.2.2, Apache Spark is an in-memory architecture, which allows intermediate datasets to be cached in distributed memory of a cluster of nodes. This characteristic prevents the intermediate datasets from being stored on hard disk. Unlike MapReduce platform, where a lot of I/O overhead is produced due to reading and writing from and to hard disk, Spark is more efficient because it avoids this overhead. For some iterative algorithm, it is advantageous.

The core of Spark is the RDDs (Resilient Distributed Datasets). Any input read from a distributed file system can be cached as a set of RDDs across a cluster of nodes. It is important to note that RDDs are immutable and are not materialized immediately. There are two types of operations for RDDs, transformation and action. A transformation function transforms RDDs from one type to another, constructing a lineage of RDDs. An action function, on the other hand, executes all the transformation functions of the lineage and returns an output, which can stored in hard disk or displayed. Both types of functions operate on many data items (contained in RDDs) concurrently.

In this chapter, we illustrate the distributed implementations of Sim_AHC and of SHCoClust, by presenting details of computing procedure, data structure, problems and solutions. To us, implementing Sim_AHC and SHCoClust is a process of "learning by doing". Though the performance of distributed Sim_AHC is not as good as expected, we gain some valuable practical programming knowledge using Spark. We share this learned knowledge in this chapter as well. For SHCoClust, like many other Spectral-SVD methods, it uses spectral embedding as core. We provide an implementation of distributed spectral embedding and make it applicable not only in SHCoClust but also in other Spectral-SVD methods. In Section 6.2 and Section 6.3, we present implementation details of distributed Sim_AHC and distributed spectral embedding, respectively.

6.2 The Distributed Implementation of Sim_AHC

6.2.1 Computing Procedure

Illustrated in Algorithm 7, having pairwise similarity matrix S, the computing procedure of Sim_AHC is composed of three steps in each iteration: firstly, searching for the pair of closest clusters; secondly, merge this pair into one cluster; and lastly, update similarity matrix S. Unlike this procedure, where computation is processed sequentially and a similarity in S can be assigned with a new value, distributed Sim_AHC needs to cache S as a set of RDDs in the distributed memory, and applies functions to update S while respecting that RDDs are immutable. The characteristic of RDDs determines that we cannot update S by assigning a new value to a similarity in it as in a conventional program. But what we can do is to create a set of new RDDs and delete a set of old RDDs. Based on this rule, we design a computing procedure in Figure 6.1. Given a collection of documents, we firstly preprocess it into a document-term matrix, then we store this matrix as a set of document vectors on HDFS. The program of distributed Sim_AHC reads from HDFS, loads these vectors and cache them as a set of RDDs in the distributed memory. After a number of iterations, the program outputs a dendrogram.

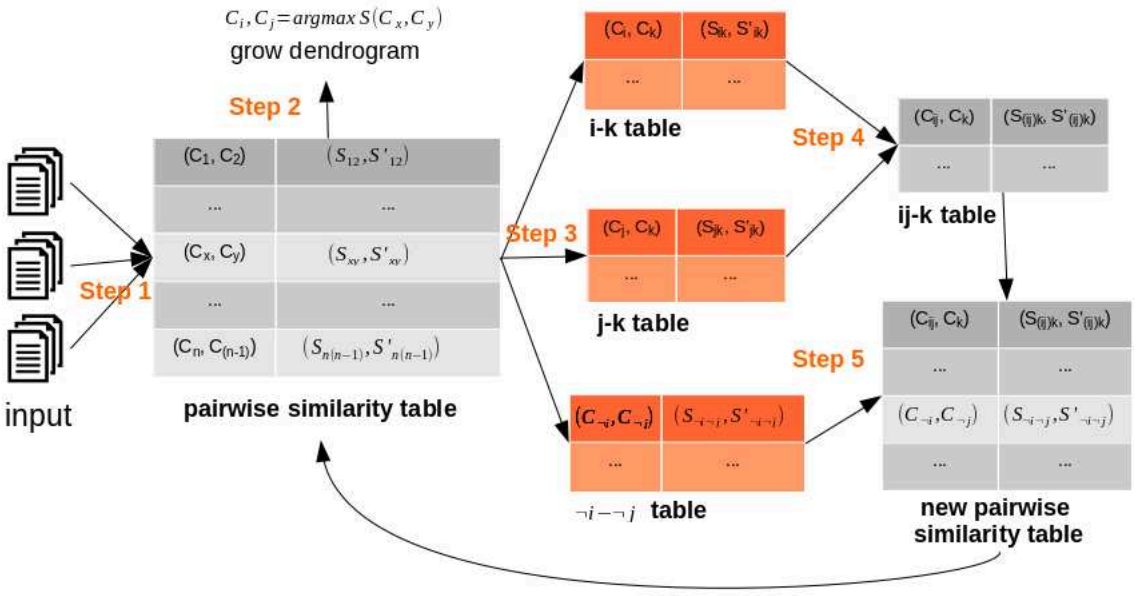


FIGURE 6.1: Computing procedure of distributed Sim_AHC using Spark RDDs. C_x and C_y denotes two clusters, S_{xy} denotes their pairwise similarity and S'_{xy} denotes their self similarity.

In details, the computing procedure of distributed Sim_AHC during each iteration is composed of five steps.

1. Firstly, our program initializes Spark RDDs by reading document vectors from HDFS. The type of the RDDs is HadoopRDD. Then all document vectors are indexed by unique ID numbers, which are associated to document names. The indexing is achieved by a map function, which converts the type from HadoopRDD into MapPartitionRDD, because individual items are in a form of key-value pair. The key is an ID number and the value is a document vector. Next, a cartesian function is applied on the MapPartitionRDD to compute pairwise similarities and self similarities. For example, given a pair of indexed document vectors such as $(C_1, [val_1, val_2, \ldots, val_m])$ and $(C_2, [val_1, val_2, \ldots, val_m])$, the output of the cartesian function on these two vectors is still in a form of key-value pair, such as $((C_1, C_2), (S_{12}, S'_{12}))$, where S_{12} is the inner product-based similarity of the two document vectors, and $S'_{12} = S_{12} - 0.5 \times [S_{11} + S_{22}]$ is their self similarity of. As document vectors are normalized during preprocessing, if an inner product is applied in the function that generates S_{12}, it is cosine similarity. This can be extended to other kernel functions. If the number of documents is n, the output of Step 1 contains n^2 pairs. By applying a filtering function (a transformation function), we reduce the number of pairs to $n(n-1)/2$. The filtered output is a set of MapPartitionRDDs, it is illustrated as a *pairwise similarity table* in Figure 6.1.

2. In the second step, our program searches for the pair of documents that has the largest pairwise similarity in the pairwise similarity table. This operation is an action function. Once it is called, all the transformation functions that are previously called on RDDs are materialized, and it outputs the key-value pair that has the maximal similarity. Let (C_i, C_j) denote the pair of documents that have the largest similarity, S_{ij}, and the corresponding self similarity is S'_{ij}. Once (C_i, C_j) is found, it is appended into a list, which is used to build a dendrogram.

3. In the third step, we firstly remove the entry $((C_i, C_j), (S_{ij}, S'_{ij}))$ from the original pairwise similarity table. The remaining entries are categorized into: (1) entries that contain index C_i in their keys, (2) entries that contain index C_j in their keys, and (3) the other entries that contain neither i nor j in their keys. Accordingly, the pairwise similarity table is split into three different tables: The i-k table, the j-k table and the $\neg i$-$\neg j$ table. C_k denotes any other remaining (document) cluster. This splitting process is achieved by three filtering functions. Recall that filter() is an RDD transformation function. The three tables are of MapPartitionRDD, whose items are cached across the cluster of nodes.

4. It is worth mentioning that entries in i-k table and in j-k table share the same set of indexes C_k in their keys. In this step, our program firstly aligns index C_k between the two tables, then applies Equation 3.11 and Equation 3.12 on the aligned pairs to update both pairwise similarities and self similarities. Depending a clustering method, the similarities are computed accordingly. The output of this step is the ij-k table. It is a UnionRDD because a union function is applied upon i-k table and j-k table.

5. In the last step, our program combines ij-k table and $\neg i$-$\neg j$ table to form a "new pairwise similarity table", which is used in the next iteration. This combination is also achieved by an union function.

In order to obtain a complete dendrogram, our program theoretically requires $n - 1$ iterations. In each iteration, it runs through Step 2-5. The program is available at `https://github.com/xywang/spark_ahc`. Note that all tables mentioned above are encapsulated as RDDs.

6.2.2 Experiments

6.2.2.1 Settings and Configurations

Our experiments are run on a cluster of five Linux machines, which are connected via a local network. We name this cluster as "minicluster". Each machine in minicluster

has 8G RAM and 4 cores. Hadoop 2.6 and Spark 2.0.0 are installed and configured on every machine. SSH communication is established. Spark standalone cluster mode is deployed, with one machine executing the driver program while the others performing executor processes. Our program is implemented in Python 2.7. In Spark, the Python API (Application Programming Interface) is often referred as PySpark[1]. PyCharm IDE (Integrated Development Environment) 4.5.1 is installed and configured on minicluster to facilitate programming.

When launching a Spark application, it is essential to specify configuration properties for the cluster, such as the number of executors, the number of executor cores, the size of executor memory, etc. There are many properties that can be configure[2], Table 6.1 lists several commonly used properties in our Spark applications. These properties can be set in Spark configuration file, or in user program, or through spark-submit command.

Property	Default	Meaning
spark.app.name	(none)	The application name.
spark.driver.cores	1	Number of cores to use for the driver process.
spark.driver.memory	1G	Amount of memory to use for the driver process.
spark.executor.memory	1G	Amount of memory to use per executor process.
spark.executor.cores	all	Number of cores to use on each executor.

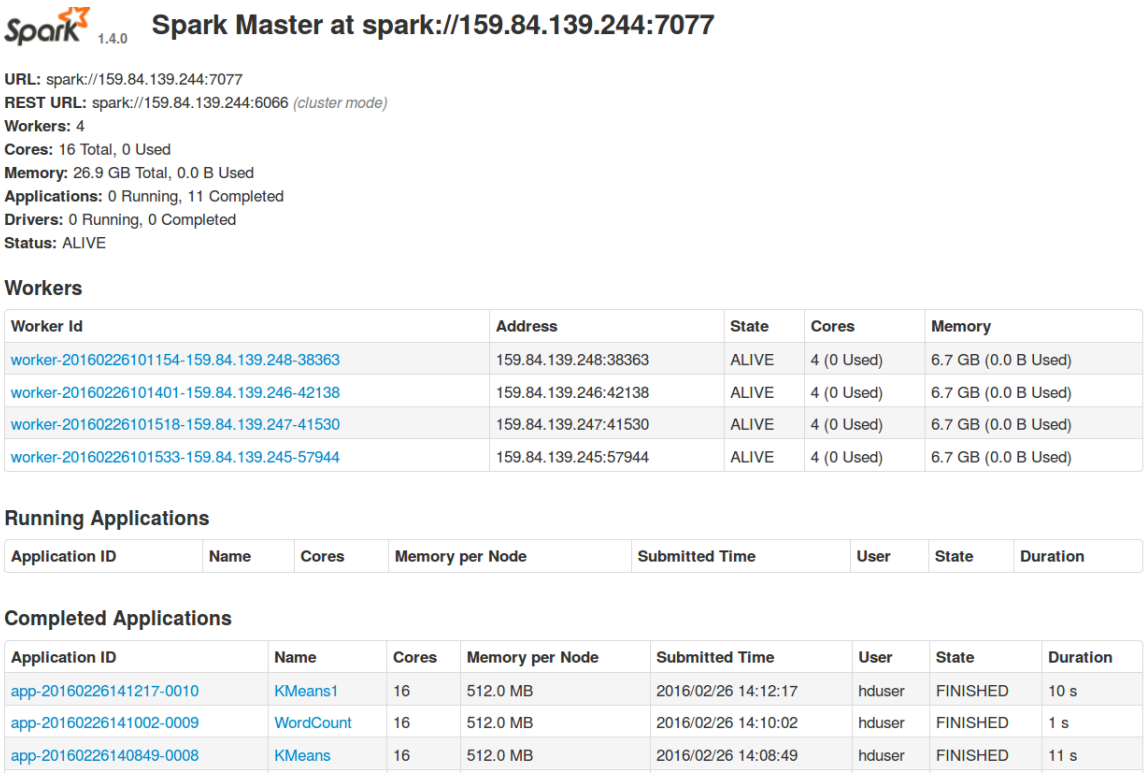
TABLE 6.1: Commonly used Spark application properties

6.2.2.2 Spark Web UI

Once launching a Spark application, SparkContext is initialized in driver program, the lineage of RDDs starts to grow by transformation functions. As these functions are not materialized immediately, it is difficult to debug a Spark program. One practical way is to observe running process from Spark web UI (User Interface). Every SparkContext launches a web UI, which is accessible by the address `http://<driver-node-ip>:4040` in a web browser. Spark web UI displays useful information of a Spark application. This includes (1) a list of scheduler jobs, stages and tasks, (2) information of running executors, (3) a summary of RDDs' sizes and memory usage and (4) environment information.

A Spark application is composed of a set of jobs, each job is further composed of a list of stages, each stage has several tasks. Figure 6.2 is a screenshot of Spark web UI homepage, which contains three parts of information: (1) cluster general information and status, (2) workers' information and status, and (3) applications' information and status. If there is an running application, we can access the list of jobs, shown in Figure 6.3. Each job is indexed by an ID number, information such as applied functions, submitted time,

[1] `https://spark.apache.org/docs/latest/api/python/index.html`
[2] `https://spark.apache.org/docs/latest/configuration.html`



Spark 1.4.0 **Spark Master at spark://159.84.139.244:7077**

URL: spark://159.84.139.244:7077
REST URL: spark://159.84.139.244:6066 *(cluster mode)*
Workers: 4
Cores: 16 Total, 0 Used
Memory: 26.9 GB Total, 0.0 B Used
Applications: 0 Running, 11 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20160226101154-159.84.139.248-38363	159.84.139.248:38363	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160226101401-159.84.139.246-42138	159.84.139.246:42138	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160226101518-159.84.139.247-41530	159.84.139.247:41530	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160226101533-159.84.139.245-57944	159.84.139.245:57944	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20160226141217-0010	KMeans1	16	512.0 MB	2016/02/26 14:12:17	hduser	FINISHED	10 s
app-20160226141002-0009	WordCount	16	512.0 MB	2016/02/26 14:10:02	hduser	FINISHED	1 s
app-20160226140849-0008	KMeans	16	512.0 MB	2016/02/26 14:08:49	hduser	FINISHED	11 s

FIGURE 6.2: Screenshot of Spark web UI homepage

duration, number of succeeded stages and number of succeeded tasks is given for each job. This information allows us to know the processing duration of different functions, and to decide which function to modify to improve efficiency. Furthermore, by clicking an active job, we can access the details of undergoing stages, shown in Figure 6.4. Similarly, each stage is indexed with an ID number, information such as applied functions, submitted time, duration, the number of succeeded tasks, input size and the size of shuffle write is present. This information allows us to have more details on bottlenecks and failures. We can also access the DAG (Directed Acyclic Graph) visualization of a job, shown in Figure 6.5. It shows the growth of lineage of RDDs through consecutive stages inside a job. It helps us to track the growth of RDDs' lineage. If the lineage is too long, it is then necessary to cut the lineage in order to save memory and to accelerate calculation.

Additionally, we can access to the information of executors, shown in Figure 6.6. This information allows us to know the status, number of RDDs' blocks, size of storage memory, number of cores, number of active, failed, complete tasks, etc for each executor. With this information, we can observe whether work load is balanced among executors.

Spark web UI provides essential information when running a Spark application, it is the most important tool to debug and improve a Spark user program.

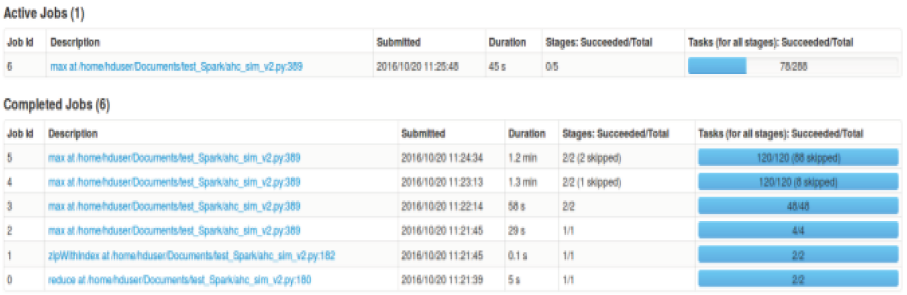


FIGURE 6.3: Screenshot of jobs of a Spark application

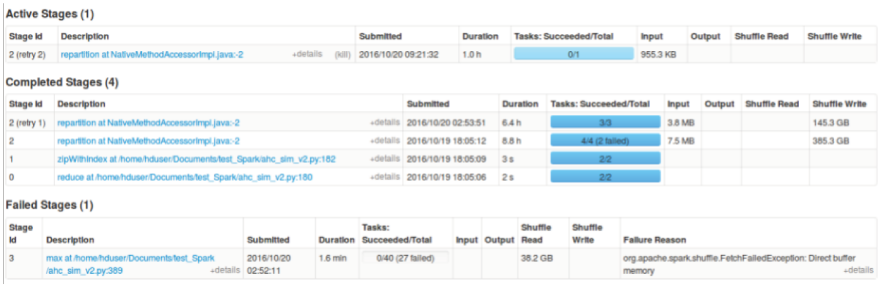


FIGURE 6.4: Screenshot of stages inside a Spark job

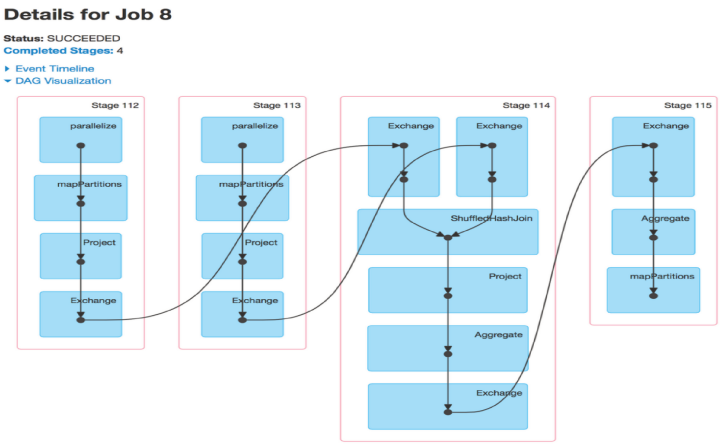


FIGURE 6.5: Screenshot of DAG of a Spark job

6.2.2.3 Exploration and Troubleshooting

Several datasets that vary in size are used in experimenting the distributed Sim_AHC implementation. These datasets are all numeric, and some of them are synthesized.

Implementing user-specified programs on Spark is exploratory and laboratory. Not only the choice of a transformation or an action function, but also a configuration property can influence the performance of the program. In our experimentation, we first test the distributed implementation on small datasets, such as Iris dataset and a few sampled datasets from x_700mb dataset and from 2cir_10xe5 dataset (the number of rows in

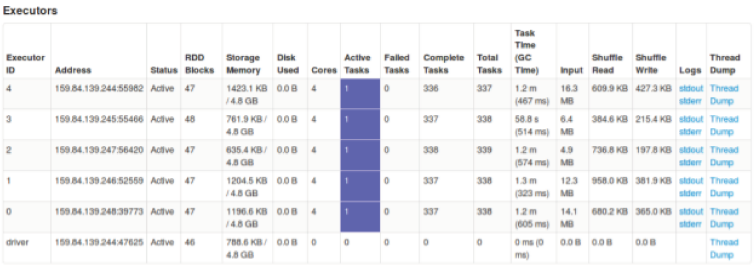


FIGURE 6.6: Screenshot of Spark executors

Dataset	No.Rows	No.Cols	Size	Type
x_700mb.txt	10,000	10,000	715Mb	synthesized
2cir_10xe5.txt	10,000	2	8.5Mb	synthesized
2cir_10xe3.txt	1,000	2	83Kb	synthesized
Iris.csv	150	4	2.2Kb	

TABLE 6.2: Datasets experimented in distributed Sim_AHC

sampled datasets vary from 100 to 300). Results obtained by these small datasets are verified against a conventional Sim_AHC implementation. The absolute value of cophenetic coefficient (CC) is used to compare two dendrograms, which are obtained from the distributed Sim_AHC implementation and from the conventional Sim_AHC implementation, respectively. In all tests that use such small datasets as input, the absolute CC value is always 1, indicating that the dendrogram obtained by the distributed Sim_AHC is identical to the dendrogram obtained by the conventional Sim_AHC. This is a positive result. However, when fed larger datasets, the program takes enormous time to return results. In this section, we present several explanations, solutions and the learned knowledge from encountered problems.

- Control over the number of partitions.

  Spark manages data using “partitions" that parallelize data processing by sending data among executors (this is referred as shuffling)[3]. A partition is a chunk of data, by default, it is created for each HDFS block, which is 128Mb[4]. Spark runs one task for one partition (the number of partitions can be seen from “Total tasks" in Spark UI). Typically we can have 2-4 partitions for each CPU in the cluster. Spark usually sets the number of partitions automatically based on the computing resource of a cluster. But this can be modified by user once a set of RDDs is created by reading from HDFS or by the parallelize() function. The number of partitions determines the level of parallelism in data processing. If it is set large for a relatively small dataset, the performance is slowed down. On the other hand,

[3] `https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd-partitions.html`
[4] `http://spark.apache.org/docs/latest/rdd-programming-guide.html`

if it is set too small for a large dataset, the computation is inefficient as well. This is a dataset-dependent parameter.

In our early experiments using Iris dataset, we artificially set the number of partitions to 2 when the data is read from HDFS. Through iterations, we observe that the number of partitions is tripled in each iteration, and processing is getting slower and slower. By tracking the number of partitions of the tables shown in Figure 6.1, we find out that the problem is caused by the union() function, which is applied in Step 4 and in Step 5. In PySpark, when applying the union() function, such as $RDD_3 \leftarrow RDD_1.\text{union}(RDD_2)$, the number of partitions of the produced RDDs is the sum of the number of partitions of the input RDDs. For example, if RDD_1 and RDD_2 each has m partitions, RDD_3 eventually has $2m$ partitions. Back to our experiment, the usage of union() in Step 4 and Step 5 results in tripled number of partitions in iteration, slowing down the process.

There are two ways to control the number of partitions, one is by repartition() function, the other one is by coalesce() function. Once passing a desired number of partitions "numPartitions", both functions are able to change the number of partitions of an input RDDs to numPartitions. The difference is that repartition() function requires to reshuffle the data in the RDDs so that it can create more or fewer partitions and balance data across the new set of partitions. This process always shuffles all data over the network[5]. As to coalesce() function, it only decreases the number of partitions to numPartitions. It avoids shuffling data, and thus is more efficient when we only want to have fewer partitions.

Due to the difference between the repartition() function and the coalesce() function, we choose to use coalesce() function to control the number of partitions after the union() function in Step 5. Specifically, we set coalesce(numPartitions=2), so that the number of tasks observed in Spark UI goes through a loop of "2-4-6-2" in each iteration. For Iris dataset, this modification makes each iteration to finish in less than 0.5s, and entire processing time is around 70s (averaged over seven clustering methods).

- Applying checkpoint to cut the lineage of RDDs.

  Since we find a way to control the number of partitions, and distributed Sim_AHC performs correctly on Iris dataset. We feed a larger dataset, 2cir_10xe3.txt to test the performance of our implementation. We observe that though the number of partitions is under control, occupied memory on each worker increases as the number of iterations increases. After the 200th iteration, all workers' memory is filled up and the program is forced to stop. In DAG visualization, we find

[5] `http://spark.apache.org/docs/latest/rdd-programming-guide.html`

that RDDs' lineage grows larger and larger as the number of iterations increases. Concretely, in one iteration, the RDDs of the pairwise similarity table (Figure 6.1) are the parent of the RDDs of i-k table, j-k table and $\neg i$-$\neg j$ table. And the child RDDs of i-k table and j-k table, the ij-k table unions with the $\neg i$-$\neg j$ table to produce a new pairwise similarity table, which is used in the next iteration. As we can see, in this process, the lineage of RDDs grows with child RDDs being used as parent RDDs between two consecutive iterations. This cycle makes the entire RDDs' lineage to grow till the end of the program. It is possible to execute on such a long lineage if the input dataset is relatively small, for example, the program works on Iris dataset. However, when the dataset gets larger, like 2cir_10xe3.txt, the lineage becomes too large to fit in the memory before the program reaches its end.

Spark is a in-memory distributed platform, it allows RDDs to be cached in the distributed memory of a cluster of nodes. The function persist() is done to do so. Besides, unwanted RDDs can be uncached to free distributed memory by unpersist() function. In our program, the input file is cached after it is read from HDFS. In order to cope the problem of lineage, the first solution that we come up with is to unpersist the RDDs of the pairwise similarity table after a number of iterations, then persist them again in a number of following iterations. This solution can be considered as a trade-off between efficiency and memory use. However, in practice, it does not improve the performance. After some trials, we find out that caching and unchaching RDDs only move them into and out of distributed memory, the growth of lineage is not affected.

Therefore we set up a checkpoint on the RDDs' lineage to cut it after a number of iterations. The checkpoint() function stores cached RDDs in the checkpoint directory on HDFS and all references of parent RDDs are removed. In our experiment that uses 2cir_10xe3 dataset, we checkpoint RDDs after every 30 iterations. In every 31^{st} iteration, checkpointed RDDs are reconstructed from checkpoint directory and used in the next 29 iterations. Though reconstructing RDDs from checkpoint directory consumes some time, overall processing proceeds correctly in our experiment. We observe that as the number of iterations increases (there are 1999 iterations for 2cir_10xe3 dataset, with number of partitions being 5), the processing time per iteration gradually reduces from 5s to 0.7s, and checkpoint reconstruction time reduces from 3.3min to 30s. It is reasonable to have this decrease in time as the size of pairwise similarity table is reduced along iterations. The whole processing time of 2cir_10xe3 dataset takes around two hours to generate a complete dendrogram.

- Approaches that try to find the maximal similarity.

To proceed our experiment, we now test on 2cir_10xe5 dataset with control on the number of partitions and RDD being checkpointed after a number of iterations. A new problem that we encounter is that the program requires endless time to find the closest pair in the very first iteration, show in Figure 6.7.

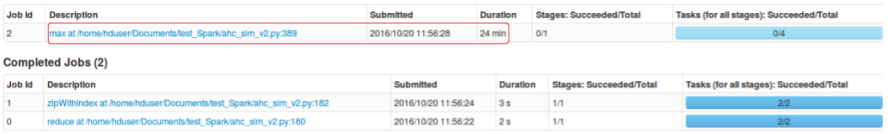


FIGURE 6.7: Screenshot of running process using 2cir_10xe5 dataset

In terms of complexity, it is of $O(n)$ to look for the maximal value in the worst case. The 2cir_10xe5 dataset has 10^5 rows, and thus $10^5 \times (10^5 - 1)/2$ pairwise similarities, among which the program has to look for the maximal. It is a very expensive operation even for a distributed system. Three solutions are tested to find feasible solutions:

1. The first solution that we provide is to take advantage of parallelism to look for the maximal similarity. Concretely, we ask the program to search for the maximal similarity in each partition. The output of this step is a list of maximal similarities, each from a partition. Then we pick the maximal value in the list. In this way, the task of searching for the maximal is shared by all partitions. However, there is an issue of this solution. When a partition is empty, error occurs because it is not possible for Spark to return any value to the driver program in this case. However, in order to verify whether a partition is empty, we have to apply an action function, which requires to materialize the full RDDs' lineage. Several different functions are programmed to achieve an efficient search. Nevertheless, it is unavoidable to use an action function in order to return a result to the driver program. This action function is expensive as it materializes the lineage, which involves generation of pairwise similarities given by the cartesian() function and searching for maximal similarity among all similarities. Even computation is performed in parallel, considering the amount of data shuffling arising among worker nodes, this task requires a lot of time to complete.

2. Sim_AHC is demonstrated its efficiency when a thresholding strategy is applied to sparsify similarity matrix. Based on this idea, our second solution is to apply such a strategy on the RDDs of pairwise similarity table. We first cache its RDDs in distributed memory and apply sparsification using a list of threshold values, each of which results in a set of child RDDs. Then we call a count() function (this is an action function) on resulting child RDDs

to observe running time. On several small samples (each has a few hundred rows) from 2cir_10xe5 dataset, we observe that as threshold value increases, the processing time of the count() function on resulting child RDDs decreases, and the number of counted values decreases as well. Similar to the count() function, RDD's max() function is also an action function, our tests imply that applying max() function on filtered RDDs would reduce the time of looking for the maximal similarity. However, if we use a threshold value to sparsify pairwise similarities, then search for the maximal in the filtered values, a question is: as pairwise similarities are updated after each iteration, how can we dynamically determine a threshold value that returns a non-empty (yet not too large) list of similarities in each iteration? Recall that it is impossible to return all similarities to the driver program when the input data is large, thus prior knowledge on the basic statistics of the similarities is unknown. In fact, in order to determine a proper threshold value, it is unavoidable to have an idea on the distribution of similarities, and this requires an action function on the whole RDDs' lineage, which is as expensive as looking for the maximal.

3. After many tests involved in the two solutions stated above, it seems that as long as the maximal similarity is searched in the full set of similarities, it is hard to avoid materializing the entire RDDs' lineage. For a dataset that has n objects, we are only interested to find the closest pair. If we are able to group the dataset into a few groups, each of which is composed of similar objects, then our objective is merely to find the closest pair in one of these groups, instead of computing all pairwise similarities and searching the maximal among them. So we formulate our problem to a problem of "looking for the nearest neighbor". A solution that we find to solve this problem is Locality Sensitive Hashing (LSH) [114]. The objective of LSH is to find the most similar pairs that are above some lower bound in similarity. The general procedure of LSH is composed of two steps: (1) it converts input data into a set of signatures, which are short but good representatives of data instances; and (2) it assigns similar data instances into different groups via their signatures. Depending on choice of similarities (such as Jaccard similarity, cosine similarity, etc), the method that computes signatures varies [115]. For example, Jaccard similarity is commonly used in searching similar web-pages, the corresponding hashing function that is used to compute the signatures of web-pages is MinHashing. Given a family of l hashing functions, $\mathcal{H} = \{h_1, \ldots, h_l\}$, each hash function operates on each data instance in the input dataset (assuming it has n instances and m features), generating a signature matrix of shape $n \times l$. This process is like projecting the input dataset into a new space of l dimensions. Each signature represents an input

data instance. Next step is to assign signatures into a number of groups, each group containing a set of similar data instances. Two more parameters are required here, the number of bands b and the number of signatures in each band r, $b \times r = l$ holds. We can consider that the signature matrix is vertically cut into b parts, each part having r columns. Hashing signatures in the signature matrix band by band and row by row outputs a matrix of $n \times b$. In the resulting matrix, if two rows are identical, then they are similar objects that are assigned into one group. Parameter $t \approx (1/b)^{1/r}$ is the threshold in LSH that defines how similar data instances have to be in order for them to be considered as a desired "similar pair". If avoidance of false negatives is important, it is better to select b and r that produce a value lower than t; if speed is important and it is desired to limit false positives, it is better to select b and r that produce a higher threshold [114].

- Python v.s. Scala.

  Spark is implemented in Scala[6], though it provides Python as an API, there is difference between a Spark Scala program and a Spark Python program in terms of performance. A Scala program runs much faster than a Python program on Spark. Python and Scala have different executing environments, Python code is interpreted by interpreter while Scala code is complied by JVM (Java Virtual Machine). In Spark, communication between a Python program and Scala functions is achieved by a wrapper package, "Py4J"[7], which enables Python programs running in a Python interpreter to dynamically access objects in a JVM. However, translation between the two different environments takes time. The performance difference is evident when a Python program is complex and requires frequent exchange with Scala functions in a Spark application.

6.2.3 Summary

In this section, we present the implementation of distributed Sim_AHC, by providing details on its computing procedure, debugging tool and practical knowledge learned from experimentation.

In looking for an answer to explain why this implementation cannot really work well on large datasets, we doubt that Spark may not be the suitable platform to an iterative algorithm like Sim_AHC, in which RDDs' lineage accumulates and only partial RDDs

[6] `https://www.scala-lang.org/`
[7] `https://www.py4j.org/`

are required to be materialized for later iterations. Recall that "Spark is for bulk iterative algorithms" [21].

The concepts of "bulk iterations" and of "incremental iterations" are distinguished in [116]: the former compute a complete new results from input data; while in the latter one, each iteration result only modifies or adds to some small subset of the input data. In the case of Sim_AHC, it is more like an incremental iterative algorithm rather than a bulk iterative algorithm, because its result obtained in each iteration is dependent on previous iteration. This process requires mutable state to be updated and carried to the next iteration. Spark is not well-optimized to address this kind of iterative algorithms. Recall that Spark RDDs are immutable, and manipulation on this property impacts computational efficiency of iterative algorithms.

6.3 The Distributed Implementation of Spectral-embedding

As seen in Algorithm 8, the core of SHCoClust is spectral embedding, which projects input data into a space that is constructed by the eigenvectors of graph Laplacian matrix. Given a document-term matrix A, performing spectral embedding on A projects it into a new space, where A becomes Z. SHCoClust applies Sim_AHC on Z, while the bipartite spectral graph partitioning method [9] applies K-means on Z. Another distributed implementation that we provide here is the distributed spectral-embedding. As spectral embedding is the basic step in Spectral-SVD methods, we believe that our implementation provides an option for other searchers to perform spectral embedding in their works.

6.3.1 Computing Procedure

Figure 6.8 illustrates the computing procedure of distributed spectral-embedding. The input is a collection of raw documents that are stored on HDFS. After preprocessing, this collection is converted into a distributed document-term matrix A, which is also store on HDFS. There are two other inputs, which are saved as two lists in the master node (where runs the driver program). They are the inverse squared root of the row sums and of the column sums of A, denoted by $D_1^{-1/2}$ and $D_2^{-1/2}$.

Differing from a conventional program, where data is cached in the RAM of a single machine. In a Spark program, an object may be cached in the RAM of the master node or in the distributed RAM of the workers. Thus a Spark program operates on both local data types and distributed data types. The Spark local vector (dense or sparse), labeled

point and local matrix belong to local data types, which can only be cached in the RAM of the master node. Data of distributed types, on the other hand, is stored as RDDs in the distributed RAM on the cluster. The RowMatrix, IndexedRowMatrix, CoordinateMatrix and BlockMatrix are distributed data types[8]. In PySpark, some functions are implemented for some data types but not for the other. For example, the SVD function is implemented for IndexedRowMatrix and for RowMatrix, but not for BlockMatrix. And the the same functions may have to be operated differently on different data types. For example, the multiply() function for IndexedRowMatrix and for RowMatrix only allows such a distributed matrix to be multiplied by a Spark local dense matrix[9]. And the multiplication function for BlockMatrix actually allows two distributed BlockMatrices to be multiplied. Luckily, in most cases, it is possible to convert a distributed matrix from one type to another.

In our implementation, two distributed data types are used, IndexedRowMatrix and BlockMatrix. We use the former type to index documents and terms in the final output, and we use latter type to perform matrix multiplication without unnecessary conversion between local and distributed data types.

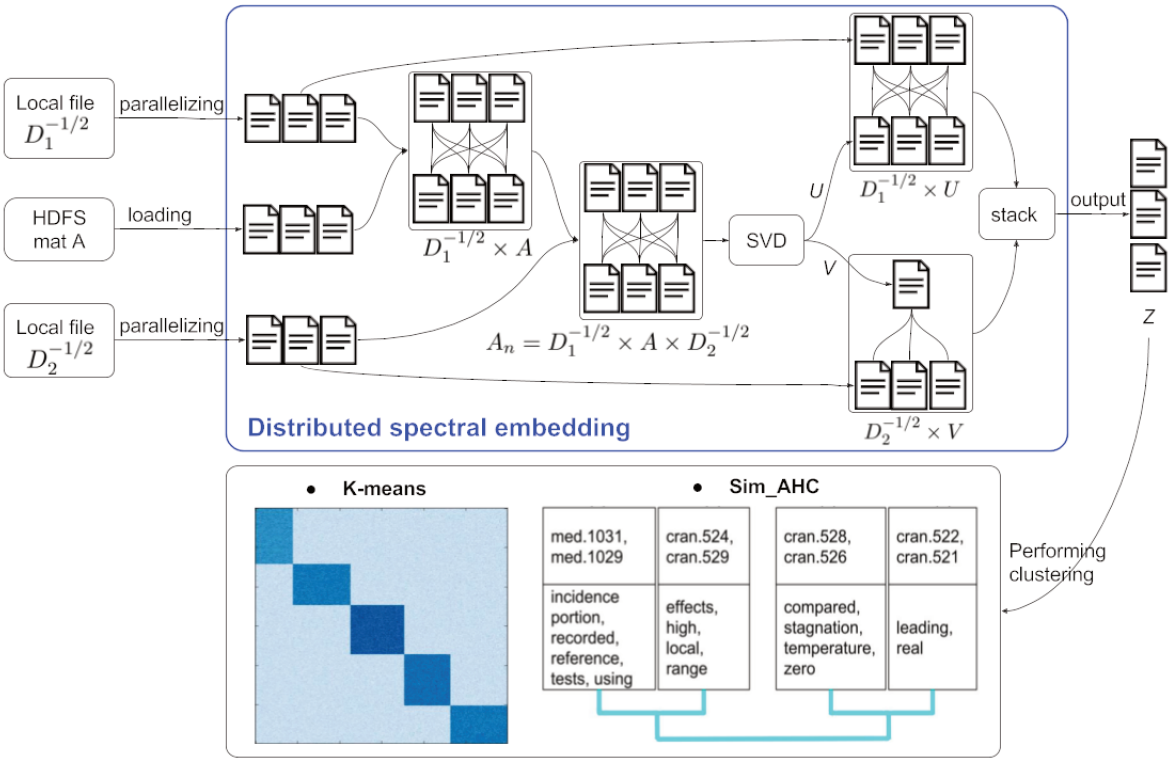


FIGURE 6.8: Computing procedure of distributed spectral embedding

[8] `https://spark.apache.org/docs/latest/mllib-data-types.html`
[9] Note that a Spark local dense matrix is different from a usual Python dense matrix.

In principle, there are fours steps to process spectral embedding in a distributed manner:

1. Loading and parallelizing inputs. As the lists of $D_1^{-1/2}$ and of $D_2^{-1/2}$ are saved locally, in the first step, they have to be loaded, parallelized and converted to distributed matrices. Concretely, after being loaded as two lists, $D_1^{-1/2}$ and $D_2^{-1/2}$ are firstly used to construct two usual sparse diagonal matrices on the master node. Then the two sparse matrices are converted to two sets of Spark sparse vectors, which are parallelized to the cluster via SparkContext. Finally, each set of parallelized vectors is converted into a distributed IndexedRowMatrix, then to a distributed BlockMatrix. Similarly, the distributed matrix A is also loaded by the SparkContext from HDFS. On HDFS, A is actually saved as a collection of row vectors, each of which is a Spark sparse vector. After it is loaded, A is converted into an IndexedRowMatrix and then to a BlockMatrix.

2. Generating matrix A_n. Shown in Algorithm 8, A_n is obtained by $D_1^{-1/2} \times A \times D_2^{-1/2}$. In this distributed implementation, A_n is generated by two multiplications. The first one is $D_1^{-1/2} \times A$ and its result is again multiplied by $D_2^{-1/2}$. Recall that A, $D_1^{-1/2}$ and $D_2^{-1/2}$ are now distributed BlockMatrices, the multiplication can be achieved by a built-in multiply() function. Consequently, A_n is also a BlockMatrix.

3. Performing SVD. After A_n is generated, it is time to apply SVD on A_n to output the matrices of the left singular vectors U and of the right singular vectors V. As mentioned previously, Spark does not yet provide a built-in SVD function for BlockMatrix. In order to apply SVD, we convert A_n to an IndexedRowMatrix. It results U, an IndexedRowMatrid and V, a Spark local dense matrix.

   Note that singular values returned by Spark's built-in SVD function are ordered decreasingly. In Algorithm 8, the first column of U and of V are removed before they are used to multiply with $D_1^{-1/2}$ and $D_2^{-1/2}$. To simply this process in this implementation, the multiplications of $D_1^{-1/2} \times U$ and $D_2^{-1/2} \times V$ are firstly performed, then the first columns of the resulting matrices are removed.

   To perform $D_1^{-1/2} \times U$, U is converted from an IndexedRowMatrix into a BlockMatrix. This multiplication results in a BlockMatrix, which is then converted into an IndexedRowMatrix. And for $D_2^{-1/2} \times V$, as V is a Spark local dense matrix, $D_2^{-1/2}$ is firstly converted from a BlockMatrix into an IndexedRowMatrix. Then the built-in multiply() function of an IndexedRowMatrix is applied. This multiplication directly outputs an IndexedRowMatrix.

   In order to remove the first columns in the resulting IndexedRowMatrices, we have to convert these two matrices into two sets of Spark indexed vectors, from which it is possible to remove elements in a specific column.

4. Generating final output. The last step is to combine the two sets of distributed indexed vectors resulted from $D_1^{-1/2} \times U$ and from $D_2^{-1/2} \times V$. The combination outputs Z, a set of distributed indexed vectors, which is stored as a set of Spark pickle files on HDFS.

After Z is output, there are different ways to perform clustering on Z. For example, we can use the distributed Sim_AHC on Z to apply (distributed) SHCoClust, we can also apply a Spark built-in clustering method, such as K-means to apply the (distributed) bipartite spectral graph partitioning method. In order to achieve efficiency, it is better to cache Z in the distributed memory during calculation.

It is worth to mention that when parallelizing a data object via SparkContext, the number of partitions is equal to the number of executor cores by default. However, in this distributed implementation, we observe that the number of partitions of Z is usually larger than the default value. This is caused by the fact that the combination of $D_1^{-1/2} \times U$ and $D_2^{-1/2} \times V$ is performed by a union() function, which automatically sums up the number of partitions of the two sets of RDDs. If Z is loaded from HDFS to perform clustering, it is advised to adjust the number of partitions of Z in order to achieve better performance.

6.3.2 Experiments and Analysis

In our experiments, we test the performance of distributed spectral embedding. Table 6.3 lists the experimented datasets, among which the SMART, AP and WSJ datasets are mentioned in Section 5.2. AP-1 and AP-2 are two sampled datasets from the full AP collection. The ZSDF dataset contains all files from four collections provided in our purchased TREC corpus[10]. These collections are "Ziff", "San Jose Mercury News", "Dept. of Energy" and "Federal Register". "Size (Mb)", "Num.Docs" and "Num.Terms" respectively list the size, the number of documents and the number of extracted terms of each dataset.

Dataset	Size (Mb)	Num.Docs	Num.Terms
START	5.8	3,893	6,812
AP-1	26.3	10,458	1,564
AP-2	53.1	20,845	1,551
WSJ	534.2	173,252	1364
AP	766.2	242,918	1540
ZSDF	1759.9	638,884	1500

Table 6.3: Experimented datasets

[10] `https://catalog.ldc.upenn.edu/LDC93T3A`

For each dataset, singular values obtained by the distributed implementation are firstly verified against those output by a conventional Python program. It is shown that the singular values obtained by the distributed and by the conventional programs are always identical. After this verification, we move to examine the performance in several scenarios, where the size of assigned executor memory and the number of total cores vary. Results of this experiment are listed in Table 6.4. "ExeMem(Gb)" indicates the size of assigned memory (measured by gigabytes) in each executor, and "Num.TotCores" is the number of total cores assigned to the cluster. We configure minicluster to have five worker nodes, i.e., one machine hosts the driver program as well as a worker program, and the other four machines only host worker programs. "Time.SVD(s)" records the running time (measured in seconds) of performing SVD, and "Time.UI(s) records the total processing time (measured in seconds) from Spark UI.

Dataset	ExeMem (Gb)	Num.TotCores	Time.SVD (s)	Time.UI (s)
SMART	1	5	14.2	34
	1	10	12.3	32
	1	15	12.4	33
AP-1	2	5	17.6	38
	2	10	18.7	40
	2	15	17.8	43
AP-2	2	5	34.4	60
	2	10	32.7	55
	2	15	23.4	46
AP	3	5	438	540
	3	10	372	468
	3	15	331	438
WSJ	6	5	234	372
	6	10	197	342
	6	15	115	216
ZSDF	6	10	405	600
	6	15	392	595
	6	20	380	582

Table 6.4: Performance of distributed spectral embedding using SMART, AP, WSJ and ZSDF datasets

When the assigned executor memory is fixed, for small datasets, it is hard to see the influence of the number of total cores on running time. As we can see in the experiments that use SMART and AP-1 datasets, the processing time for SVD and the total running time first decrease then increase as more cores are assigned. As to larger datasets, some trend can be observed. It is clear that in the experiments of AP-2, AP, WSJ and ZSDF datasets, as the number of total cores increase, the processing time of SVD and the total running time both decrease.

One explanation is that when too many cores are used to process a relatively small dataset, the benefit of distributed computing is lessened by the overhead of network communication and the cost of data shuffling among executors. However, when a dataset is large enough for a number of assigned cores and the amount of assigned memory, the overhead of network communication and the cost data shuffling become less dominant compared to the amount of processing time. As result, the time gain of distributed computing becomes more evident.

The results of this experiment imply that our implementation is capable to scale, when the number of cores increases, the processing time decreases. However, there is a condition for this scalability, i.e., the number of cores assigned in processing should be suitable to the size of input dataset. Otherwise, the computation cannot benefit from being distributed.

One of the bottlenecks of this implementation is the Spark built-in SVD() function. This function is implemented in Java and wrapped with py4j package so that it can be used as a Python function. However, it is not possible to adjust the level of parallelism inside this function through Python. By observing the running process, we think that the number of tasks is fixed in this function, and it cannot automatically adapt to the number of assigned cores. For a relatively large dataset, this can be problematic as the available computing power cannot be utilized, and cached data has to wait for being processed. Another bottleneck is the BlockMatrix, which cannot yet accept a really large dataset. And it does not support operations on sparse matrix either.

6.4 Conclusion

In this section, we present two distributed implementations, the distributed Sim_AHC and the distributed spectral embedding. For both implementations, we illustrate details on their computing procedures, debugging tool, data types and configuration parameters. Implementing a Spark program is different from implementing a conventional program. Configuration parameters, such as the number of cores, the size of executor memory, etc., have important impact on the performance of a Spark application. And there is no rule of thumb to set optimal values to these parameters. The only solution is to learn them from experiments. In implementing the distributed Sim_AHC, we obtain valuable experience from many failures. Though this program is being made better through many troubleshooting trials, it is still not performing as expected for a real large dataset. So far it is capable to return accurate results on small and medium-sized datasets. What is more important is our experience learned from this progress, such as the control over the number of partitions, the necessity of cutting an endlessly growing lineage by checkpoint,

application of LSH to find similar objects, etc. Some of these learned knowledge provide us valuable practice in implementing the distributed spectral embedding and other programs using Spark. For SHCoClust, we implement its core, the spectral embedding, in a distributed manner. The key point of this implementation is to correctly convert data types between local types and distributed types, and to choose suitable distributed types when data is presented in a distributed matrix. Though there exist some bottlenecks due to some Spark built-in functions, this distributed implementation works properly. In our experiments, we demonstrate its scalability on several datasets that vary in size. We believe that this implementation provides an option in performing spectral embedding for methods that apply spectral embedding.

Chapter 7

Conclusions and Perspectives

7.1 Conclusions

In this thesis, with addressing text clustering tasks as our focus, we elaborate on our contributions on toward scalable hierarchical clustering methods, tests on the cluster hypothesis and distributed implementations.

First of all, interested in organizing documents in a collection and in preserving their interconnections, we study hierarchical clustering methods. Our focus is the conventional AHC methods that are unified by the Lance-Williams formula. The problem of conventional AHC methods is that they are computationally costly, due to their high complexity. To overcome this drawback, we propose Sim_AHC, a similarity-based hierarchical framework. It is equivalent to the Lance-Williams formula, but it uses inner product-based similarities instead of distances. This characteristic makes Sim_AHC advantageous: (1) as its similarities are between 0 and 1, we can apply a thresholding strategy to sparsify its similarity matrix. This results in improved computing efficiency. In our experiments, we find out that sparsification indeed leads to reduced running time and memory use. And surprisingly, clustering quality is preserved and even improved. Our analysis is that by sparsifying similarity matrix, noise is removed. Furthermore, it connects neighborhoods and absorbs data points that are close to these neighborhoods. (2) using inner product-based similarities can easily extend similarities in Sim_AHC to kernel functions. In our experiments, we perform tests using similarity matrix that is generated by both linear and Gaussian kernels. In the work of Sim_AHC, our initial objective is to provide an equivalent framework to the Lance-Williams formula, with better computing efficiency.

Secondly, co-clustering methods draw our attention with its capability of simultaneously clustering data instances and data features. This property is beneficial in text

clustering, as it can return co-clusters of documents and terms. However, in common co-clustering methods, there is no information on how resulting co-clusters are connected and how elements in a co-cluster is structured. This motivates us to propose SHCoClust, a similarity-based hierarchical co-clustering method. It is a hybrid algorithm that processes the characteristic of co-clustering and hierarchical clustering. But it is more advantageous than the two. One one hand, the output of SHCoClust has richer information. It outputs a dendrogram, but unlike a dendrogram that is output by a hierarchical clustering, its dendrogram aggregates both documents and terms. By cutting this dendrogram, we can obtain a number of sub-dendrograms, each of which is a co-cluster that organizes its elements in a hierarchy. On the other hand, SHCoClust also uses inner product-based similarities. Like Sim_AHC, it can be extended to different kernel functions. And more importantly, a thresholding strategy can be applied in SHCoClust to achieve better computing efficiency. In our experiments, we discover that clustering quality of SHCoClust is guaranteed or improved when sparsification is applied. Comparing with conventional AHC methods, clustering quality in SHCoClust is much improved. More importantly, we find that when sparsification is applied, on average 75% running time and memory use can be spared, while preserving clustering quality.

Thirdly, after applying Sim_AHC and SHCoClust in text clustering tasks, we further apply them in testing the cluster hypothesis. We believe that by testing this hypothesis, we are able to have better understanding on retrieval effectiveness and efficiency of these two frameworks. Another interest of doing this work is to provide a benchmark on this topic. In reviewing past research works, we find that in terms of which clustering method achieves the best retrieval effectiveness, there are different conclusions. Besides, efficiency issue is not sufficiently discussed in these works. In our tests that applies Sim_AHC and SHCoClust, we use optimal cluster search and the E-measure to evaluate retrieval effectiveness. In comparing retrieval effectiveness among seven clustering methods, we conclude that Ward method is the most effective method in most cases in the test of Sim_AHC. However, in the test of SHCoClust, a wider range of performing methods is obtained. In terms of efficiency, we examine the influence of improving efficiency by sparsifying similarity matrix on retrieval effectiveness. We discover that in both tests retrieval effectiveness tends to be invariant to the effect of sparsification, with substantial memory use and running time being reduced. Our analysis is that sparsifying similarity matrix like disconnects clusters that are near the root of a dendrogram, but leaves little impact on optimal clusters that are usually near the leaf nodes of the dendrogram. And in our experiments, we find that most optimal clusters are small clusters that are near the leaf nodes. In comparing Sim_AHC and SHCoClust, we discover that in MED, CISI and CACM datasets, Sim_AHC is more effective than SHCoClust using complete link, average link, McQuitty and Ward methods. On the contrary, SHCoClust is more

effective than Sim_AHC using single link, centroid and median methods in AP and WSJ datasets. Besides, as SHCoClust generates more flat clusters than Sim_AHC, it has smaller variance of optimal E values than Sim_AHC.

Lastly, we provide distributed implementations of Sim_AHC and SHCoClust using Apache Spark engine. Along our study of Spark and implementing Sim_AHC and the core of SHCoClust, spectral embedding, we learn that the performance of a Spark program depends not only on the implementation, but also on the configuration of computing resources. After many trials and tests, we manage to implement the distributed Sim_AHC and the distributed spectral embedding. Along this process, we gain useful practical experience of using Spark and we share this experience in this thesis. We hope this can be a helpful reference for other people that are interest in implementing similar algorithms.

7.2 Perspectives

There are some points that we like to improve in our works: first of all, sparsification used in Sim_AHC and SHCoClust depends on a threshold value. Usually we pass a list of threshold values to see which one gives the best clustering quality with the least computing resources. But in practice, we probably want a function that can determines such a threshold value automatically. On this subject, we have not examined sufficiently. And this is one of the further works that we would like to contribute. Secondly, in testing the cluster hypothesis using Sim_AHC and CoClust, we feel that there is more to do in examining optimal clusters. During our experiments, we observe that the size of optimal cluster usually links to its location in a dendrogram and affects the value of precision and recall. But we do not have systematic knowledge on this matter. It would be worth examining the link between the size of optimal cluster to retrieval effectiveness in the future. Lastly, as stated previously, we doubt that Spark may not be the suitable platform for Sim_AHC. An alternative might be using a parallel distributed computing platform. And we would like to look into this direction. We also notice that Spark is gradually moving to DataFrame from RDDs. Maybe we can use DataFrame to improve the implementation of distributed spectral embedding.

Appendix A

Publication List

International Conferences

1. Xinyu Wang, Julien Ah-Pine, and Jerome Darmont. Shcoclust, a scalable similarity-based hierarchical co-clustering method and its application to textual collections. In IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 17), Naples, Italy, July 2017.

2. Julien Ah-Pine and Xinyu Wang. Similarity based hierarchical clustering with an application to text collections. In International Symposium on Intelligent Data Analysis, pages 320331. Springer, 2016.

National Conferences

1. Xinyu Wang, Julien Ah-Pine, and Jerome Darmont. A new test of cluster hypothesis using a scalable similarity-based agglomerative hierarchical clustering framework. In Rencontres Jeunes Chercheurs en Recherche dInformation (CORIA 17), Marseille, March 2017.

2. Julien Ah-Pine and Xinyu Wang. Classification ascendante hiérarchique à noyaux et pistes pour un meilleur passage à léchelle. In Journées de Statistique de la SFDS, Lille, France, 2015.

3. Julien Ah-Pine and Xinyu Wang. Classification ascendante hiérarchique à noyaux et une application aux données textuelles. In EGC, volume vol.RNTI-E-33 of EGC 2017. Revue des Nouvelles Technologies de lInformation, Grenoble, France, 2017.

Appendix B

Résume

Comme une méthode d'apprentissage automatique non supervisé, la classification automatique est largement appliquée dans des tâches diverses. Différentes méthodes de la classification ont leurs caractéristiques uniques. La classification hiérarchique, par exemple, est capable de produire une structure binaire en forme d'arbre, appelée dendrogramme, qui illustre explicitement les interconnexions entre les instances de données. Le co-clustering, d'autre part, génère des co-clusters, contenant chacun un sous-ensemble d'instances de données et un sous-ensemble d'attributs de données.

Dans cette thèse, nous travaillons sur des données textuelles. Compte tenu d'un corpus de documents, nous adoptons l'hypothèse de «bag-of-words» et applique le modèle vectoriel. Nos données saisies sont transformées à une matrice de document-terme, qui est remplie de poids TF-IDF. L'avantage de regrouper des documents à l'aide du regroupement hiérarchique est qu'il organise des documents et ne nécessite pas de nombre du groupes prédéfinis. Cependant, la procédure du calcul est coûteuse en raison d'une haute complexité. Dans le cadre de cette thèse, nous travaillons sur les techniques classiques de classification ascendante hiérarchique et nous proposons le Sim_AHC [7]. C'est une expression de la formule de Lance et Williams [26] en fonction de produits scalaires plutôt qu'en termes de distances. Nous établissons les conditions dans lesquelles cette nouvelle expression est équivalence à la méthode initiale. L'intérêt de cette approche est double. Tout d'abord, nous pouvons étendre naturellement les techniques classiques de classification ascendante hiérarchique aux fonctions noyaux. Ensuite, le raisonnement sur des matrices de produits scalaires est davantage propice à la définition de méthodes de seuillage de mesures de proximités. Nous proposons alors de pré-traiter la matrice de proximités de façon à la rendre éparse afin de permettre un meilleur passage à l'échelle de ces techniques de classification.

Avec la formule de Lance et Williams et les méthodes classiques de classification ascendante hiérarchique comme notre base de référence, nos expériences utilisant des proximités générées par le noyau gaussien et le noyau linéaire démontrent que, les résultats obtenus par Sim_AHC sont identiques à ceux produits par les méthodes initiales. Plus important encore, lorsque la matrice de proximités est rendue plus en plus éparse, l'utilisation de la mémoire et du temps de fonctionnement diminuent. Par contre, la qualité de la classification est garantie, grâce au fait que les bruits sont supprimés par le seuillage.

Contrairement à la classification hiérarchique, le co-clustering effectue la classification dans l'espace de données et l'espace des attributs. Cependant, le co-clustering ne peut pas préserver l'interconnexion des éléments qu'il regroupe. Pour surmonter cet inconvénient, nous proposons SHCoClust [8], la méthode de co-clustering hiérarchique basée sur la similarité. Il est considéré comme une méthode hybride de Sim_AHC et de co-clustering spectral. Concrètement, dans SHCoClust, nous modelons un corpus de documents comme une graphe bipartite, dont les sommets sont des documents et des termes. Ensuite, nous appliquons la méthode de spectral-SVD [63] pour couper la graphe en plusieurs sous-graphes, chaque étant un co-cluster. La méthode de spectral-SVD, en fait, construit une espace avec les vecteurs propres de la matrice laplacienne de la graphe bipartite. Puis, elle projette la matrice originale dans cette espace. Nous appliquons la classification hiérarchique Sim_AHC sur la matrice transformée.

SHCoClust hérite des caractéristiques de la classification hiérarchique Sim_AHC et du co-clustering spectral. Il produit un dendrogramme composé à la fois de documents et de termes. En coupant le dendrogramme, nous pouvons obtenir un certain nombre de co-clusters, dont chacun est un co-cluster hiérarchique, c'est-à-dire que les interconnexions de documents et de termes dans un co-cluster sont préservées. Plus important encore, comme SHCoClust utilise également des proximités du produit scalaire, nous pouvons également l'étendre aux fonctions du noyau et nous pouvons appliquer une stratégie du seuillage pour rendre la matrice éparse. Nos expériences démontrent que la qualité de la classification de SHCoClust est en grande partie améliorée par rapport aux méthodes de la classification hiérarchique conventionnelle. Par rapport à la méthode de co-clustering spectral, SHCoClust réalise une amélioration lorsque sa matrice de proximités est rendu éparse. En outre, nous constatons que, en épargnant la matrice de proximités, bien que moins de mémoire et moins de temps soient nécessaires pour effectuer le calcul, la qualité de la classification peut être garantie. Dans nos ensembles de données testés, en moyenne, un gain du mémoire et un gain du temps jusqu'à 75 % sont obtenus sans nuire à la qualité de la classification.

L'hypothèse de cluster [92] est une hypothèse fondamentale pour les applications de la recherche d'informations basées sur la classification automatique. Il indique que les documents dans le même groupe ont tendance à être pertinents pour la même requête. En testant cette hypothèse, il nous permet de examiner comment une requête est répondu. De nombreux travaux [93, 94, 95, 98, 100] passés effectuent des tests sur cette hypothèse, en utilisant des méthodes de la classification hiérarchique conventionnelle. Certains de ces travaux vérifient si l'hypothèse de cluster répond à un ensemble de données testé. Certains comparent des stratégies de recherche d'information dans un dendrogramme. Autres examinent lequel méthode de la classification hiérarchique donne la meilleur efficacité de recherche.

Cependant, les conclusions des travaux par rapport à la méthode de la classification la plus efficace ne sont pas cohérentes, en raison des différences dans les mesures d'évaluations, les paramètres expérimentaux et les ensembles de données testés. En outre, l'efficacité du calcul n'est pas discutée. Puis, seulement quartes méthodes de la classification hiérarchique conventionnelle sont testés. Intéressés à fournir une référence mise à jour et plus complète pour les tests de la l'hypothèse de cluster, nous proposons deux nouveaux tests en appliquant les méthodes proposées, le Sim_AHC [119] et le SHCoClust. Pour chaque méthode, nous obtenons d'abord des dendrogrammes générés par les méthodes de la classification. Ensuite, nous utilisons l'E mesure pour évaluer l'efficacité de la recherche sur un dendrogramme. L'E mesure est une mesure impartiale qui calcule la moyenne harmonique de la précision et du rappel. Une valeur haute signifie une bonne efficacité de recherche. L'utilisation de l'E mesure est dans le contexte de la recherche de cluster optimale, qui permet de trouver le cluster le plus pertinent à une requête dans un dendrogramme. Dans nos expériences, nous utilisons l'E mesure pour testr et comparer l'efficacité des méthodes de la classification dans le Sim_AHC et dans le SHCoClust. Puis, pour examiner l'efficacité du calcul, nous aussi testons l'influence de rendre la matrice de proximités éparse sur l'efficacité de la recherche, et nous appliquons une test statistique pour comparer les résultats obtenus par le Sim_AHC et par le SHCoClust.

Par rapport à la méthode la plus efficace, nos expériences utilisant des proximités générées par le noyau linéaire et le noyau gaussien montrent que la méthode du lien moyen et la méthode de Ward sont les méthodes les plus efficaces lors de l'utilisation de Sim_AHC. Cependant, lors de l'utilisation de SHCoClust, les méthodes les plus efficaces deviennent le lien simple, le lien moyen, le centroïde, Ward et McQuitty. Nos résultats sont partiellement d'accord avec des découvertes dans les travaux passés sur le même sujet. En termes d'influence de rendre la matrice du proximités éparse sur l'efficacité de recherche, nous constatons que l'efficacité a tendance à être invariante. En fait, les valeurs de l'E mesure se gardent presque au même niveau, même si la matrice de proximités est rendu plus en plus éparse. C'est un résultat intéressant. Il signifie que c'est possible d'avoir la

même efficacité de recherche en utilisant beaucoup moins de mémoire et de temps pour effectuer le calcul.

En comparant les résultats de la test du Sim_AHC et de la test du SHCoClust à l'aide d'un test de Student, nous découvrons que Sim_AHC est plus efficace que SHCoClust lorsqu'il utilise des méthodes de lien simple, lien complet, lien moyen, McQuitty et Ward dans de petits ensembles de données. Cependant, SHCoClust est plus efficace que Sim_AHC en utilisant des méthodes de lien simple, lien moyen et centroïde dans des ensembles de données relativement plus grandes.

Intéressés par effectuer le calcul pour des ensembles de données vastes, nous choisissons l'Apache Spark[1] pour implémenter Sim_AHC et SHCoClust. Le Spark est une plate-forme du calcul distribué. Il utilise la capacité du calcul collective d'un groupe de noeuds pour traiter des ensembles de données vastes. En général, le Spark fonctionne sur un système de fichier distribué, qui est établi sur un groupe de machines. Après une initialisation, le Spark coupe des tâches du calcule et les assigne aux noeuds, qui ensuite effectuent leurs tâches simultanément. Le concept de base du Spark est les données distribuées résilientes (RDDs) [21]. Comme une abstraction grossier, une RDD est en fait un morceau de données qui est mit en mémoire-cache distribué. Le RDD est immuable et il y a deux groupes de fonctions qui peuvent traiter RDDs. Ces sont les fonctions de transformation et d'actions. Les fonctions de transformation permet aux RDDs de croître en une forme de lignée, et les fonctions d'actions coupe la lignée de RDDs, effectue le calcul et renvoie les résultats. La caractéristique de RDD demande une façon différente que les programme conventionnels en terme d'implémentation.

Bien que le Spark gère automatiquement la planification des travaux, la réplication des données et la communication réseau parmi les noeuds, il existe encore de nombreux paramètres à régler afin d'optimiser l'efficacité et l'évolutivité du calcul, par exemple, le nombre de morceaux des RDDs, le nombre de tâches à assigner et la taille de mémoire-cache à utiliser dans chaque noeud. Il est aussi important de contrôler la longueur de la lingée des RDDs. Nous fournissons deux implémentations distribuées, le Sim_AHC distribué et la méthode de spectral-SVD distribuée. Pour chaque implémentation, nous illustrons la procédure et la performance du calcul. Nous trouvons que l'implémentation du Sim_AHC distribué ne fonctionne pas aussi bien que prévu. Après avoir essayé des solutions différentes, nous concluons que le Spark est peut-être pas une plate-forme du calcul appropriée pour un algorithme comme Sim_AHC, dans lequel une itération dépend l'itération précédente. Le Spark est plutôt un bon choix pour les algorithmes qui fonctionnent aux itérations indépendantes par lot. Dans cette thèse, nous partageons nos connaissances savantes de nos expériences, en croyant qu'elles seraient utiles pour

[1] `https://spark.apache.org/`

d'autres chercheurs, qui s'intéressent à la mise en oeuvre de la méthode hiérarchique à l'aide de Spark. Pour l'implémentation distribuée de SHCoClust, nous proposons une façon distribuée de réaliser la méthode de spectral-SVD. Dans nos expériences, nous utilisons des données de tailles différentes pour examiner la caractéristiques de l'échelle.

Mots-clés : classification ascendante hiérarchique, co-clustering, recherche d'informations, l'hypothèse de cluster, calcul distribué.

Bibliography

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[2] Guojun Gan, Chaoqun Ma, and Jianhong Wu. *Data clustering: theory, algorithms, and applications*. SIAM, 2007.

[3] Yongkweon Jeon and Sungroh Yoon. Multi-threaded hierarchical clustering by parallel nearest-neighbor chaining. *IEEE Transactions on Parallel and Distributed Systems*, 26(9):2534–2548, 2015.

[4] Michel Marie Deza and Elena Deza. Encyclopedia of distances. In *Encyclopedia of Distances*, pages 1–583. Springer, 2009.

[5] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.

[6] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *ACM Sigmod Record*, volume 25, pages 103–114. ACM, 1996.

[7] Julien Ah-Pine and Xinyu Wang. Similarity based hierarchical clustering with an application to text collections. In *International Symposium on Intelligent Data Analysis*, pages 320–331. Springer, 2016.

[8] Xinyu Wang, Julien Ah-Pine, and Jerome Darmont. Shcoclust, a scalable similarity-based hierarchical co-clustering method and its application to textual collections. In *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 17), Naples, Italy*, July 2017.

[9] Inderjit S Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–274. ACM, 2001.

[10] Charu C Aggarwal and ChengXiang Zhai. A survey of text clustering algorithms. In *Mining text data*, pages 77–128. Springer, 2012.

[11] Douglass R Cutting, David R Karger, Jan O Pedersen, and John W Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 318–329. ACM, 1992.

[12] Florian Beil, Martin Ester, and Xiaowei Xu. Frequent term-based text clustering. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 436–442. ACM, 2002.

[13] Oren Zamir and Oren Etzioni. Web document clustering: A feasibility demonstration. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 46–54. ACM, 1998.

[14] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

[15] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[16] Charu C Aggarwal, Stephen C Gates, and Philip S Yu. On using partial supervision for text categorization. *IEEE Transactions on Knowledge and data Engineering*, 16(2):245–255, 2004.

[17] A Blum and T Mitchell. Learning to classify text from labeled and unlabeled documents. In *Conference on Computational Learning Theory*, 1998.

[18] Xiang Ji and Wei Xu. Document clustering with prior knowledge. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 405–412. ACM, 2006.

[19] Ron Bekkerman, Mikhail Bilenko, and John Langford. *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press, 2011.

[20] Mohammed J Zaki and Ching-Tien Ho. *Large-scale parallel data mining*. Springer Science & Business Media, 2000.

[21] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[22] Grant Ingersoll. Introducing apache mahout. *Scalable, commercial-friendly machine learning for building intelligent applications. IBM*, 2009.

[23] William Gropp. Tutorial on mpi: The message-passing interface. *Mathematics and Computer Science Division Argonne National Laboratory Argonne, IL*, 60439, 2009.

[24] Blaise Barney. Openmp tutorial. *Lawrence Livermore National Laboratory, https://computing. llnl. gov/tutorials/openMP/# Abstract*, 2011.

[25] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems*. Prentice-Hall, 2007.

[26] Godfrey N Lance and William Thomas Williams. A general theory of classificatory sorting strategies: Ii. clustering systems. *The computer journal*, 10(3):271–277, 1967.

[27] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2:86–97, 2012.

[28] Robin Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The computer journal*, 16(1):30–34, 1973.

[29] F James Rohlf. Hierarchical clustering using minimum spanning tree, 1973.

[30] Daniel Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 1977.

[31] C De Rham. La classification hiérarchique ascendante selon la méthode des voisins réciproques. *Les cahiers de l'analyse des données*, 5(2):135–144, 1980.

[32] J Juan. Programme de classification hiérarchique par l'algorithme de la recherche en chaîne des voisins réciproques. *Les cahiers de l'analyse des données*, 7(2):219–225, 1982.

[33] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983. doi: 10.1093/comjnl/26.4.354.

[34] Michel Bruynooghe. Méthodes nouvelles en classification automatique de données taxinomiques nombreuses. *Statistique et Analyse des données*, 2(3):24–42, 1977.

[35] Michael Rex Anderberg. *Cluster analysis for applications*. PhD thesis, Office of the Assistant for Study Support, 1972.

[36] Roberto J López-Sastre, Daniel Oñoro-Rubio, Pedro Gil-Jiménez, and Saturnino Maldonado-Bascón. Fast reciprocal nearest neighbors clustering. *Signal Processing*, 92(1):270–275, 2012.

[37] Bastian Leibe, Krystian Mikolajczyk, and Bernt Schiele. Efficient clustering and matching for object class recognition. In *BMVC*, pages 789–798, 2006.

[38] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. In *ACM SIGMOD Record*, volume 27, pages 73–84. ACM, 1998.

[39] Yaniv Loewenstein, Elon Portugaly, Menachem Fromer, and Michal Linial. Efficient algorithms for accurate hierarchical clustering of huge datasets: tackling the entire protein space. *Bioinformatics*, 24(13):i41–i49, 2008.

[40] Yijun Sun, Yunpeng Cai, Li Liu, Fahong Yu, Michael L Farrell, William McKendree, and William Farmerie. Esprit: estimating species richness using large collections of 16s rrna pyrosequences. *Nucleic acids research*, 37(10):e76–e76, 2009.

[41] Thuy-Diem Nguyen, Bertil Schmidt, and Chee-Keong Kwoh. Sparsehc: a memory-efficient online hierarchical clustering algorithm. *Procedia Computer Science*, 29: 8–19, 2014.

[42] S Shalom, Manoranjan Dash, and Minh Tue. An approach for fast hierarchical agglomerative clustering using graphics processors with cuda. *Advances in Knowledge Discovery and Data Mining*, pages 35–42, 2010.

[43] Zhihua Du and Feng Lin. A novel parallelization approach for hierarchical clustering. *Parallel Computing*, 31(5):523–527, 2005.

[44] William Hendrix, Md Mostofa Ali Patwary, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary. Parallel hierarchical clustering on shared memory platforms. In *High Performance Computing (HiPC), 2012 19th International Conference on*, pages 1–9. IEEE, 2012.

[45] Shen Wang and Haimonti Dutta. Parable: A parallel random-partition based hierarchical clustering algorithm for the mapreduce framework. *Technical Report CCLS-11 04*, 2011.

[46] Chen Jin, Zhengzhang Chen, William Hendrix, Ankit Agrawal, and Alok Choudhary. Incremental, distributed single-linkage hierarchical clustering algorithm using mapreduce. In *Proceedings of the Symposium on High Performance Computing*, pages 83–92. Society for Computer Simulation International, 2015.

[47] Chen Jin, Ruoqian Liu, Zhengzhang Chen, William Hendrix, Ankit Agrawal, and Alok Choudhary. A scalable hierarchical clustering algorithm using spark. In *Big Data Computing Service and Applications (BigDataService), 2015 IEEE First International Conference on*, pages 418–426. IEEE, 2015.

[48] Gérard Govaert and Mohamed Nadif. *Co-clustering*. John Wiley & Sons, 2013.

[49] Inderjit S Dhillon, Subramanyam Mallela, and Dharmendra S Modha. Information-theoretic co-clustering. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 89–98. ACM, 2003.

[50] Sara C Madeira and Arlindo L Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 1(1):24–45, 2004.

[51] Amos Tanay, Roded Sharan, and Ron Shamir. Biclustering algorithms: A survey. *Handbook of computational molecular biology*, 9(1-20):122–124, 2005.

[52] Guandong Xu, Yu Zong, Peter Dolog, and Yanchun Zhang. Co-clustering analysis of weblogs using bipartite spectral projection approach. *Knowledge-Based and Intelligent Information and Engineering Systems*, pages 398–407, 2010.

[53] Thomas George and Srujana Merugu. A scalable collaborative filtering framework based on co-clustering. In *Data Mining, Fifth IEEE international conference on*, pages 4–pp. IEEE, 2005.

[54] Arindam Banerjee, Inderjit Dhillon, Joydeep Ghosh, Srujana Merugu, and Dharmendra S Modha. A generalized maximum entropy approach to bregman co-clustering and matrix approximation. *Journal of Machine Learning Research*, 8 (Aug):1919–1986, 2007.

[55] Gérard Govaert and Mohamed Nadif. Clustering with block mixture models. *Pattern Recognition*, 36(2):463–473, 2003.

[56] Gerard Govaert and Mohamed Nadif. An em algorithm for the block mixture model. *IEEE Transactions on Pattern Analysis and machine intelligence*, 27(4): 643–647, 2005.

[57] Gérard Govaert and Mohamed Nadif. Fuzzy clustering to estimate the parameters of block mixture models. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 10(5):415–422, 2006.

[58] Gérard Govaert and Mohamed Nadif. Clustering of contingency table and mixture model. *European Journal of Operational Research*, 183(3):1055–1066, 2007.

[59] Gérard Govaert and Mohamed Nadif. Block clustering with bernoulli mixture models: Comparison of different approaches. *Computational Statistics & Data Analysis*, 52(6):3233–3245, 2008.

[60] Gérard Govaert and Mohamed Nadif. Latent block model for contingency table. *Communications in StatisticsTheory and Methods*, 39(3):416–425, 2010.

[61] Gilles Celeux, Didier Chauveau, and Jean Diebolt. Stochastic versions of the em algorithm: an experimental study in the mixture case. *Journal of Statistical Computation and Simulation*, 55(4):287–314, 1996.

[62] Malika Charrad, Yves Lechevallier, Mohamed Ben Ahmed, and Gilbert Saporta. On the number of clusters in block clustering algorithms. In *FLAIRS Conference*, 2010.

[63] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[64] Lars Hagen and Andrew B Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE transactions on computer-aided design of integrated circuits and systems*, 11(9):1074–1085, 1992.

[65] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[66] Dorothea Wagner and Frank Wagner. Between min cut and graph bisection. *Mathematical Foundations of Computer Science 1993*, pages 744–750, 1993.

[67] Hongyuan Zha, Xiaofeng He, Chris Ding, Horst Simon, and Ming Gu. Bipartite graph partitioning and data clustering. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 25–32. ACM, 2001.

[68] Manjeet Rege, Ming Dong, and Farshad Fotouhi. Bipartite isoperimetric graph partitioning for data co-clustering. *Data Mining and Knowledge Discovery*, 16(3): 276–312, 2008.

[69] Bojan Mohar, Y Alavi, G Chartrand, and OR Oellermann. The laplacian spectrum of graphs. *Graph theory, combinatorics, and applications*, 2(871-898):12, 1991.

[70] Bojan Mohar. Some applications of laplace eigenvalues of graphs. In *Graph symmetry*, pages 225–275. Springer, 1997.

[71] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[72] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[73] Michael Holmes, Alexander Gray, and Charles Isbell. Fast svd for large-scale matrices. In *Workshop on Efficient Machine Learning at NIPS*, volume 58, pages 249–252, 2007.

[74] Zhaojun Bai, James Demmel, Jack Dongarra, Axel Ruhe, and Henk van der Vorst. *Templates for the solution of algebraic eigenvalue problems: a practical guide*. SIAM, 2000.

[75] Stephen Guattery and Gary L Miller. On the quality of spectral separators. *SIAM Journal on Matrix Analysis and Applications*, 19(3):701–719, 1998.

[76] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2001.

[77] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788, 1999.

[78] Tao Li and Chris HQ Ding. Nonnegative matrix factorizations for clustering: A survey., 2013.

[79] Chris Ding, Tao Li, Wei Peng, and Haesun Park. Orthogonal nonnegative matrix t-factorizations for clustering. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 126–135. ACM, 2006.

[80] Hua Wang, Feiping Nie, Heng Huang, and Fillia Makedon. Fast nonnegative matrix tri-factorization for large-scale data co-clustering. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1553, 2011.

[81] Chris Ding, Xiaofeng He, and Horst D Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 606–610. SIAM, 2005.

[82] Spiros Papadimitriou and Jimeng Sun. Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 512–521. IEEE, 2008.

[83] Tugdual Sarazin, Mustapha Lebbah, and Hanane Azzag. Biclustering using spark-mapreduce. In *BigData Conference*, pages 58–60, 2014.

[84] Sen Su, Xiang Cheng, Lixin Gao, and Jiangtao Yin. Co-clusterd: a distributed framework for data co-clustering with sequential updates. In *2013 IEEE 13th International Conference on Data Mining*, pages 1193–1198. IEEE, 2013.

[85] Yanfeng Zhang, Qixin Gao, Lixin Gao, and Cuirong Wang. imapreduce: A distributed computing framework for iterative computation. *Journal of Grid Computing*, 10(1):47–68, 2012.

[86] Lynda Tamine and Laure Soulier. Collaborative information retrieval: Concepts, models and evaluation. In *Advances in Information Retrieval - 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20-23, 2016. Proceedings*, pages 885–888, 2016. doi: 10.1007/978-3-319-30671-1_86. URL `https://doi.org/10.1007/978-3-319-30671-1_86`.

[87] Lynda Tamine and Laure Soulier. Collaborative information retrieval: Frameworks, theoretical models, and emerging topics. In *Proceedings of the 2016 ACM on International Conference on the Theory of Information Retrieval, ICTIR 2016, Newark, DE, USA, September 12-6, 2016*, pages 7–8, 2016. doi: 10.1145/2970398.2970442. URL `http://doi.acm.org/10.1145/2970398.2970442`.

[88] Anagha Kulkarni and Jamie Callan. Selective search: Efficient and effective search of large textual collections. *ACM Transactions on Information Systems (TOIS)*, 33(4):17, 2015.

[89] Yubin Kim, Jamie Callan, J Shane Culpepper, and Alistair Moffat. Efficient distributed selective search. *Information Retrieval Journal*, 20(3):221–252, 2017.

[90] Gilad Katz, Anna Shtock, Oren Kurland, Bracha Shapira, and Lior Rokach. Wikipedia-based query performance prediction. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 1235–1238. ACM, 2014.

[91] Hadas Raviv, Oren Kurland, and David Carmel. Query performance prediction for entity retrieval. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 1099–1102. ACM, 2014.

[92] Nick Jardine and Cornelis Joost van Rijsbergen. The use of hierarchic clustering in information retrieval. *Information storage and retrieval*, 7(5):217–240, 1971.

[93] Ellen M Voorhees. The cluster hypothesis revisited. In *Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 188–196. ACM, 1985.

[94] W Bruce Croft. A model of cluster searching based on classification. *Information systems*, 5(3):189–195, 1980.

[95] Abdelmoula El-Hamdouchi and Peter Willett. Techniques for the measurement of clustering tendency in document retrieval systems. *Journal of Information Science*, 13(6):361–365, 1987.

[96] Cornelis Joost Van Rijsbergen and W Bruce Croft. Document clustering: An evaluation of some experiments with the cranfield 1400 collection. *Information Processing & Management*, 11(5):171–182, 1975.

[97] Oren Kurland. The cluster hypothesis in information retrieval. In *European Conference on Information Retrieval*, pages 823–826. Springer, 2014.

[98] Alan Griffiths, Lesley A Robinson, and Peter Willett. Hierarchic agglomerative clustering methods for automatic document classification. *Journal of Documentation*, 40(3):175–205, 1984.

[99] Peter Willett. Recent trends in hierarchic document clustering: a critical review. *Information Processing & Management*, 24(5):577–597, 1988.

[100] Alan Griffiths, H Claire Luckhurst, and Peter Willett. Using interdocument similarity information in document retrieval systems. *Readings in Information Retrieval, Morgan Kaufmann Publishers, San Francisco, CA*, pages 365–373, 1997.

[101] Mark D Smucker and James Allan. A new measure of the cluster hypothesis. In *Conference on the Theory of Information Retrieval*, pages 281–288. Springer, 2009.

[102] Anastasios Tombros. *The effectiveness of query-based hierarchic clustering of documents for information retrieval*. PhD thesis, University of Glasgow, 2002.

[103] Fiana Raiber and Oren Kurland. The correlation between cluster hypothesis tests and the effectiveness of cluster-based retrieval. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 1155–1158. ACM, 2014.

[104] Fiana Raiber and Oren Kurland. Exploring the cluster hypothesis, and cluster-based retrieval, over the web. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2507–2510. ACM, 2012.

[105] ChengXiang Zhai. Statistical language models for information retrieval. *Synthesis Lectures on Human Language Technologies*, 1(1):1–141, 2008.

[106] M. Bruynooghe. Classification ascendante hiérarchique des grands ensembles de données : un algorithme rapide fondé sur la construction des voisinages réductibles. *Cahiers de l'analyse des données*, 3(1):7–33, 1978.

[107] Nello Cristianini and John Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[108] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[109] Ken Lang. NewsWeeder : learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 331–339, 1995.

[110] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[111] Aaron F McDaid, Derek Greene, and Neil Hurley. Normalized mutual information to evaluate overlapping community finding algorithms. *arXiv preprint arXiv:1110.2515*, 2011.

[112] Abdelmoula El-Hamdouchi and Peter Willett. Comparison of hierarchic agglomerative clustering methods for document retrieval. *The Computer Journal*, 32(3): 220–227, 1989.

[113] Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[114] A Rajaraman and JD Ullman. Finding similar items. *Mining of Massive Datasets*, 77:73–80, 2010.

[115] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014.

[116] Stephan Ewen, Kostas Tzoumas, Moritz Kaufmann, and Volker Markl. Spinning fast iterative data flows. *Proceedings of the VLDB Endowment*, 5(11):1268–1279, 2012.

[117] Julien Ah-Pine and Xinyu Wang. Classification ascendante hiérarchique à noyaux et pistes pour un meilleur passage à l'échelle. In *Journées de Statistique de la SFDS*, Lille, France, 2015. URL `https://hal.archives-ouvertes.fr/hal-01504642`.

[118] Julien Ah-Pine and Xinyu Wang. Classification ascendante hiérarchique à noyaux et une application aux données textuelles. In *EGC*, volume vol.RNTI-E-33 of *EGC 2017. Revue des Nouvelles Technologies de l'Information*, Grenoble, France, 2017. URL `https://hal.archives-ouvertes.fr/hal-01525446`.

[119] Xinyu Wang, Julien Ah-Pine, and Jerome Darmont. A new test of cluster hypothesis using a scalable similarity-based agglomerative hierarchical clustering framework. In *Rencontres Jeunes Chercheurs en Recherche d'Information (CORIA 17), Marseille*, March 2017.