Md. Delwar Hossain
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Master of Computer Science
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Querying Communities of Interest in Peer Database Networks
TITRE DE LA THÈSE / TITLE OF THESIS

Iluju Kiringa
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Sivarama Dandamudi

Paola Flocchini

Gary W. Slater
LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURS ET POSTDOCTORALES / DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
The Thesis of Md. Delwar Hossain is approved:

__________________________________

__________________________________

__________________________________

Committee Chairperson

University of Ottawa, Canada
QUERYING COMMUNITIES OF INTEREST IN
PEER DATABASE NETWORKS

by

Md. Delwar Hossain

THESIS
Submitted in partial fulfillment of the requirements for the
degree of Master of Science in Computer Science

UNIVERSITY OF OTTAWA
July 2005
NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.
ACKNOWLEDGMENTS

I am grateful to some people for the successful completion of this research. Without each of them, this work would have been very difficult to accomplish.

First, I would like to thank my thesis Supervisor, Prof. Dr. Kiringa for his support during this work, his continuous guidance, the countless hours he spent discussing ideas or reviewing my work, and all the fruitful thought that laid the foundation of the research presented in this thesis.

I also thank Mujtaba Khambatti, for supplying his materials and answering lot of questions over e-mail in spite of his busy schedule. His idea helped me proper directions to build our research interest.

Thanks to Anastasios Kementsietsidis for providing his thesis materials. I am thankful to the University of Ottawa for providing the computer laboratory. I am also grateful to some of university staff and colleagues for ample enthusiasm towards my work.

Last, I would like to thank my family members for providing me with courage and support during this period.
DEDICATION

In the name of Allah, the most gracious.

To my daughters

Samiha and Raya.
ABSTRACT

Peer-to-Peer (P2P) networks are used primarily for file sharing. P2P user communities are continuing to grow rapidly, but there is no specific query mechanism for communities of interest in peer database networks. Peer databases are linked to each other through acquaintances. These are individual, independently developed databases that contain local data. Many researchers have tackled the problem of query processing in P2P networks. Moreover, researchers have started to investigate community-based querying in peer database networks. The focus of this thesis is to study a query translation mechanism, to develop an algorithm for querying communities of interest in peer database networks, and to implement a prototype of this algorithm.

P2P communities are formed using common claimed items. First, we investigate an existing community formation and discovery algorithm for a text file, as well as an existing query translation mechanism for peer databases. After that, we develop a community-based search algorithm for peer databases. The developed algorithm combines the aforementioned community formation and discovery with the query translation mechanisms.

A prototype has been developed and experimental results are shown. An approach for discovering communities ‘on the fly’ is introduced and a method for optimizing the community discovery technique is shown.
# TABLE OF CONTENTS

List of Tables ........................................................................ vii
List of Figures ........................................................................ viii

CHAPTER

1 Introduction ......................................................................... 1
  1.1 Synopsis ........................................................................ 1
  1.2 Contributions ............................................................... 2
  1.3 Organization ................................................................. 3

2 Background .......................................................................... 5
  2.1 Database Systems .......................................................... 5
     2.1.1 Federated and Multi-database Systems ...................... 5
     2.1.2 Peer-to-Peer Database Systems ............................... 6
     2.1.3 P2P Information Systems ...................................... 8
     2.1.4 Data Mapping / Integration in P2P Systems ............. 9
  2.2 P2P Query Processing ...................................................... 11
     2.2.1 P2P Search Techniques .......................................... 11
     2.2.2 Query Translation and Propagation ......................... 11
  2.3 P2P Communities ......................................................... 12
     2.3.1 Web Communities ................................................ 12
     2.3.2 P2P Community Formation and Discovery ............. 14
     2.3.3 Community-based Search Techniques ..................... 16
  2.4 Summary and Conclusions .............................................. 18
3 Querying Communities of Interest

3.1 Community-based Querying Method

3.1.1 Basic Definitions

3.2 Motivating Example of Community-based Search

3.2.1 Peer Schemas and Database Graph

3.2.2 ItemFile of Schemas

3.2.3 Attribute Mappings

3.2.4 Mapping Tables

3.2.5 Running Example

3.2.6 Working Procedure of Our Example

3.3 Algorithm for Community-based Search

3.4 Discussion of Our Algorithm

3.5 System Architecture of Community-based Search

4 Implementation and Experiments

4.1 Motivation

4.2 Khambatti’s Community Discovery Technique

4.3 Optimizing Khambatti’s Community Discovery Technique

4.4 Benefit of Modified Technique

4.5 Discovering Communities ‘on the fly’

4.5.1 Communities ‘on the fly’

4.5.2 Algorithm for Discovering Communities ‘on the fly’

4.5.3 Discussion

4.6 Implementation
4.7 Experiments.................................................................54

4.7.1 Community Refreshment within Different Time Intervals .............55

4.7.2 Effect of Query Propagation for Community Changes ...............57

4.7.3 Percentage of Communities Discovered for Different Sizes of Peer Networks .........................................................58

4.7.4 Complexity of Our Algorithm ........................................59

5 Conclusion and Future Work ...............................................64

5.1 Conclusion.................................................................64

5.2 Future Work ............................................................65

BIBLIOGRAPHY.....................................................................66

APPENDIX

A Sample Outputs 70
List of Tables

3.1 Attribute mapping between P₁ and P₇
3.2 Attribute mapping between P₇ and P₁₀
3.3 Mapping table between YearToMovieYear
3.4 Mapping table between MovieYearToYearOfMovie
3.5 Mapping table between TypeIDToMovCategory
3.6 Mapping table between CountryNameToCountry
3.7 Mapping table between CountryToCountryOfMovie
List of Figures

3.1 Database graph of peer $P_1$
3.2 Database graph of peer $P_7$
3.3 Database graph of peer $P_{10}$
3.4 ItemFile with schema link of peer $P_1$
3.5 ItemFile with schema link of peer $P_7$
3.6 ItemFile with schema link of peer $P_{10}$
3.7 Diagram of movie community
3.8 Architecture for a community-based search of a P2P system
4.1 Community refreshment by changing peers/nodes.
4.2 Community refreshment by changing attributes/items
4.3 Effect of community changes on the completion time a given query: SELECT MovieID, Year FROM movie WHERE MovieID="M110" OR Year="00"
4.4 Effect of community changes on the completion time a given query: SELECT MovieID, Year FROM movie WHERE MovieID="M110" AND Year="00"
4.5 Effect of communities discovered for different sizes of networks
4.6 Community formation and link weight calculation
4.7 Community discovery
4.8 Community search of a given query and query translation
4.9 Screen-shot of the GUI of declared claimed items
4.10 Screen-shot of query generation, translation, and community discovery
Chapter 1

Introduction

1.1 Synopsis

Peer-to-Peer (P2P) computing is defined as the distribution of computer resources and information through direct exchange. It is a tool highly in demand because it does not need dedicated servers. A node is any single computer connected to a network. A node can work as a client as well as a server. Efficient P2P search techniques are proposed in [16, 39]. Moreover, a search mechanism based on the idea of communities of interest is proposed in [Khambatti, 2003]. A database management system (DBMS) is a set of software programs that controls the organization, storage and retrieval of data in a database. It also controls the security and integrity of the database. Meanwhile, DBMSs are being built around the P2P computing paradigm [2,10, 33]. Such DBMSs are called peer DBMSs. However, there is no specific query mechanism for communities of interest in networks of peer DBMSs.

The term “P2P community” comes from common interests. Here, common interests mean the same claimed items. “A non-empty set N of peers forms a peer-to-peer community if all peers in N claims the same attributes” [Khambatti, 2003].

The issue is two fold: (1) how to discover and form communities; (2) to apply community-based search algorithms. Khambatti et al. [23, 24] describe an algorithm for forming and discovering communities. They mention that extensive computation or
communication on the components of a peer is not required. Community-based querying assumes guidelines on how users will search in an interest-based locality.

Active research is under way regarding P2P database management. Most of this research (e.g., Piazza project, Hyperion project, etc) is still theoretical. However, some of the research provides prototype implementations. The Hyperion project [2] develops algorithms for peer database management. Peer DBMSs form a network of nodes (peers) which coordinate querying, updating, and sharing of data at run-time. One of the most important features of a network of peer DBMSs is that peers establish and abolish links at run-time. In this context, the set of all peer DBMSs breaks down into multiple communities of interest formed around common, shared attributes. This thesis provides an algorithm for community-based query processing in peer DBMSs. The idea is to exploit the concept of community of interest to enhance query processing in a network of peer DBMSs.

The key assumptions of our solution are: pre-defined threshold values, items/attributes assigned to each node, each node belonging to one pre-determined group (e.g., AOL group, IBM group, etc), and manually created mapping tables.

1.2 Contributions

We consider a community-based search technique for query processing in peer database networks. The main contributions are:

1. To form and discover communities, extending techniques proposed in [Khambatti, 2002].
2. To combine the community formation and discovery algorithms with a query processing algorithm developed in [22].

3. To consider both explicit communities and community discovery 'on the fly'. 'On the fly' refers to execution time.

Our algorithm forms peer communities by using an attribute escalation method which consists in having a peer claim as many attributes as possible. After discovering the formed communities, our algorithm uses a query translation technique to get results from other peers in targeted communities. Our contribution is the development of a query processing algorithm that will translate a query posed against a peer to the vocabulary of all the peers that belong to the same communities of interest. We implement explicit communities. Moreover, we show how to adapt our framework to discovering communities 'on the fly'.

1.3 Organization

This thesis is organized as follows:

Chapter 2 reviews related and background work. Here, we review federated and multi-database systems, as well as work done on peer-to-peer databases systems, P2P search techniques, and previous uses of the concept of community (e.g., web communities, community formation and discovery).

Chapter 3 discusses our community-based search algorithms. This chapter introduces a pseudo-code describing the algorithms and gives our system architecture. The chapter also explains our approach with a motivating example.
Chapter 4 presents an optimization of Khambatti’s community discovery procedures. It also contains our experimental results and discusses how communities may be discovered ‘on the fly’. Finally, the chapter discusses experimental results obtained from a prototype that we implemented.

Chapter 5 concludes our work with recommendations for future directions.
Chapter 2

Background

2.1 Database Systems

This section describes federated and multi-database systems, Peer-to-Peer (P2P) database systems, P2P information systems, and data mapping/integration in P2P systems.

2.1.1 Federated and Multi-database Systems

A federated database system (FDBS) is a collection of cooperating but autonomous component database systems (DBSs) [35]. The software that provides controlled and coordinated manipulation of the component DBSs is called a federated database management system (FDBMS). Federated database systems are mainly of two types. The first one is loosely coupled (e.g. MRDSM) and the second one is tightly coupled. Tightly coupled federated database systems have single federation or multiple federations. Sheth and Larson [35] describe federated database systems that have global schema and satisfy distribution, heterogeneity, and autonomy characteristics. For developing FDBSs, Sheth and Larson introduce a reference architecture that has data, databases, commands, processors, schemas, and mapping components.

Litwin et al. [29] review multi-database systems that have no global schema. The participant databases of a multi-database system are inter-operable. A set of (multi) databases for which a multi-database language exists is called a multi-database. The
queries of multi-databases are intended to be reusable and probably even remain valid despite changes in the schema of some of the accessed databases. The multi-database language MSQL (Mini Structured Query Language) introduced in this paper has thirteen different new features. The most important of these features is that multiple autonomous databases are able to cooperate in the processing of multi-database queries.

2.1.2 Peer-to-Peer Databases Systems

Peer-to-peer database (P2P DB) networks are a well-substantiated option for the next generation of multi-database systems. Databases can be kept and controlled in local peers or can be made autonomous for other peers. Autonomy in this context means that peers decide themselves how to develop their databases, what DBMS to use, how to store data, etc. Such a database kept on a peer is called a peer database.

In Giunchiglia [10], data coordination for interacting among peer databases is proposed. The authors introduce four basic notions to that end: interest groups, acquaintances, coordination rules, and correspondence rules. Interest groups are sets of nodes that are able to answer queries about a certain topic. Acquaintances are recognizable nodes that a node knows about and that have data that can be used to answer a specific query. Coordination rules specify under what conditions - when, how and where - to propagate queries or updates. Correspondence rules take care of the semantic heterogeneity problem. They are implemented as rewrite rules and are called by coordination rules, in the body of the code implementing their action and condition components.
Arenas et al. [2] present an architecture of peer database management systems (PDBMS) that has three main components: an interface (P2P API), a P2P layer, and a database. The P2P API is the user interface to the whole system. The P2P layer is the key component of the whole system and comprises three sub-components: an Acquaintance Manager (AM), a Query Manager (QM), and a Rule Manager (RM). The AM is accountable for semi-automatically establishing acquaintances among PDBMSs. The QM executes two types of queries: local queries and global queries. A local query is executed using only the data in the local peer, while a global query uses the P2P network to complement or reconcile locally-retrieved data with data that reside in other peers. The RM enforces stabilization and consistency policies between peers. It creates two sets of rules. The first set includes ones that manage consistency between the data of two peers. The second set of rules is created at query time, after an acquaintance has been established. Gribble [13] emphasizes the complexities of data placement and retrieval as well as the affecting components of data placement problems.

In [3,34], a data model called Local Relational Model (LRM) is proposed. The LRM assumes that the set of all data in a P2P network consists of local relational databases. Each local relational database has a set of acquaintances that defines the P2P network topology. The LRM layer has four modules: User Interface (UI), Query Manager (QM), Update Manager (UM), and Wrapper. The UI allows a user to define queries, receive results and messages from other nodes, and control other modules of the P2P layer. The QM and the UM are responsible for queries and update propagation. They manage domain relations, coordination formulas, coordination rules, acquaintances, and
interest groups. The Wrapper provides a translation layer between the QM and UM and the local data source.

In [14], a Unified Peer-to-Peer Database Protocol (UPDP) is proposed. The UPDP includes messaging, communication, and network protocol models. The Peer-to-Peer Database Protocol (PDP) has a number of key properties. It can interact with any node topology (e.g. ring, star, graph, and tree) and with multiple P2P response modes. Hoschek [14] emphasizes maintaining and querying dynamic and timely information services, resources, and user communities. He provides abstract and concrete PDP messaging models. He uses QUERY, RECEIVE, INVITE, and CLOSE request messages and a SEND response message in the abstract PDP messaging model. For the concrete PDP messaging model, he uses the abstract PDP messaging model with an extensible exchange protocol. The messages QUERY, RECEIVE, INVITE, CLOSE, SEND of the abstract PDP messaging model are refined in a concrete PDP messaging model. The concrete PDP messaging model contains three request message types, two reply message types, one answer message type, and the ERR error type.

2.1.3 P2P Information Systems

Several P2P infrastructures for file sharing exist, such as Napster, Gnutella, Freenet, Chord, JXTA and Gridella. Several of these infrastructures are described in [1]. However, effort is increasingly being spent on using these P2P infrastructures to build full fledged information systems. Such information systems find application in e-commerce and distributed data management. In this section, we briefly describe some of the early P2P infrastructures and information systems that were built on top of a P2P infrastructure, namely Kademlia, PeerDB, and Chord.
Kademlia [32] uses XOR-based metric topology. It is a P2P <key, value> storage and lookup system. It minimizes the number of configuration messages nodes must send to learn about each other. Kademlia can send a query to any node within an interval, allowing it to select routes based on latency or even send parallel asynchronous queries. To locate nodes near a particular ID, Kademlia uses a single routing algorithm from start to finish.

PeerDB [36] is a peer-to-peer distributed data sharing system that has content-based searching facilities. PeerDB is a database application implemented on top of BestPeer, a generic P2P system designed to serve as a platform on which P2P applications can be developed easily and efficiently.

The problem of locating documents in P2P systems is identified in [5] whose authors propose a P2P file sharing system based on Chord. Hashing is a common technique of speeding up access to a collection of data. It transforms a string of characters into a usually shorter fixed-length value or key that represents the original string. Hashing is used in large databases to speed up searching. Chord is an efficiently distributed lookup system based on consistent hashing. It is not a storage system. It associates keys with nodes rather than with data values.

2.1.4 Data Mapping / Integration in P2P Systems

In [20] the problem of mapping data in P2P systems is described. P2P systems rely on value searches to locate data of interest, but different peers can use different values to recognize the same data. To deal with this, P2P systems may rely on mapping tables which record pairs of corresponding values for search domains that are used in
different peers. The mapping tables are binary tables containing pairs of corresponding identifiers from two different sources and are used for searching data. The authors of [20] mention the semantic and algorithmic issues of using mapping tables and present a language that allows the user to specify mapping tables under different semantics. They also provide a solution to the problems of checking whether a set of mapping tables is consistent and whether a mapping table may be inferred from a set of given mapping tables.

Madhavan et al. [31] first present a powerful framework for defining languages for specifying mappings and their related semantics. The authors consider an instance of structure for a language representing mappings between relational data and present sound and complete algorithms for the corresponding inference problems.

Lenzerini [28] mentions a strong connection between query answering in data integration and query answering in database with complete information under constraints. The author discusses several approaches to data integration: LAV (local-as-view), GAV (global-as-view), and GLAV (approach that mixes LAV and GAV). The sources defined in terms of the global schema are called source-centric or local-as-view or LAV. The global schema is defined in terms of the sources called global-schema-centric, or global-as-view, or GAV. Lenzerini also shows how to use the GLAV approach in P2P systems where no global schema is assumed. The differences between GAV, LAV, GLAV, and P2P systems are reflected in the sort of formulae used to capture data integration. Finally, Lenzerini presents several open problems with respect to data integration including P2P data integration, data cleaning, reasoning on queries and views, optimization, etc.
2.2 P2P Query Processing

2.2.1 P2P Search Techniques

P2P search techniques are the most vital part of P2P networks. A number of P2P search techniques are available. This section reviews some of them, namely directed Breadth-First-Search (BFS), modified BFS, iterative deepening, local indices, and Intelligent Search Mechanism (ISM). These are the efficient P2P search techniques in comparison with other existing P2P search techniques. Modified BFS is an extension of the current Gnutella protocol which allows searching with keywords and is designed to minimize the number of messages that are needed to search the networks. ISM uses the past behavior of the P2P networks to further improve the scalability of the search procedure. It builds a profile of peers and uses that profile to find for each query which peers are likely to answer the query. It then forwards the query to those peers only.

2.2.2 Query Translation and Propagation

Query translation and propagation are described in [6, 22, 30, 40]. Kementsietsidis and Arenas [22] present a new mechanism of query translation in peer database systems using mapping tables. They introduce T-queries (tabular form of data) which are used in their query translation algorithm. This query translation mechanism is used for getting answers from the other peers. The authors also provide algorithms for testing whether a given query is a sound translation of another given query and define the notions of sound and complete translations. Sound translations only retrieve correct answers whereas complete translations retrieve all correct answers and no incorrect answers.
Doucet and Lumineau [30] describe the evolution of communities by introducing community knowledge in the process of query propagation. Here, knowledge is used to select the nodes for doing queries. Knowledge also means a user belonging to a community, or establishing a link between related communities. The authors define communities as a small set of keywords that characterize a field of interest. The query first propagates the relevance-based links. At the same time, the query propagates following inter-community links. Then the query linearly propagates towards relevant nodes for finding relevant resources. This process repeats until the end of the Time to Live (TTL) of the query. TTL is an 8-bit field in the Internet Protocol (IP) header that indicates how many more hops this packet should be allowed to make before being discarded or returned. Hops are a number of nodes the data must pass through before it reaches its destination. This value is used to determine the most efficient route. For authoritative records the TTL is fixed at a specific length.

2.3 P2P Communities

P2P communities are formed around common interest attributes. This section reviews some of the notions of community used in the literature, along with related algorithms and search techniques.

2.3.1 Web Communities

Flake et al. [7] describe methods for identifying web communities and develop an approximate discovery method that works well in practice. Both the ideal (the entire web can be used for the calculation) and the approximate methods (the authors use this
method for experiment) are used for identifying web communities; they are then compared. Web communities can be defined as a set of web pages that link in either direction to more web pages in the community than to pages outside of the community. The "Max flow-min cut" theorem shows that the maximum flow of the network is identical to the minimum cut that separates a source written to the community and a target outside of it. If we provide a small number of seed web pages, then a small subset of a community can be identified. For this reason, a method for identifying new seeds named Expectation Maximization (EM) algorithm is introduced. In this method, newly discovered web sites are relabeled as seeds, then by re-crawling from the new seeds a new graph is induced. The process reiterates after returning the maximum flow procedure. A maximum-flow based focused crawler can identify new web communities.

By analyzing the link topology, Gibson et al. [9] develop a notion of hyperlinked communities on the WWW. They develop an experimental system with the concepts of Hyperlink Induced Topic Search (HITS) for the purpose of studying communities on the web. The HITS algorithm uses the notions of 'hub' and 'authority'. A 'Hub' is a page that links many pages of the same topic. An 'Authority' is a page that is linked to many other pages of the same topic. The HITS algorithm takes a user's keyword query describing some 'topic' as an input, and gives N most-authoritative sites on that 'topic' as an output. Web communities are extracted through link topology using the HITS algorithm.

Kaykova et al. [19] emphasize the formation processes of global P2P networks of web-based resources using a mechanism called OntoShell. The OntoShell mechanism is used in a service-oriented environment where each service component has a semantic
profile in the form of a Resource Description Framework (RDF) graph based on the corresponding ontology. The purpose of the formation of P2P communities is to provide services in term of industrial assets and resource management systems. The service provider puts services on OntoShell and creates special meeting places to find partners to get services.

2.3.2 P2P Community Formation and Discovery

The community formation and discovery techniques are described in [23, 24, 38]. In the approach of Khambatti et al. [23], communities are formed using claimed interest attributes in active or passive ways. In the active way, claimed attributes are advertised to the ‘neighbour’ peer from a given peer; in the passive way, a peer advertises its interest attributes on its website and relies on the active approaches of other peers. The process which is used for forming communities is called attribute escalation. The set of attributes of a peer is divided into personal and claimed attributes. Personal attributes are private to a peer. In the attribute escalation process, a peer moves an attribute from the set of personal attributes to the set of claimed attributes whenever it knows that the attribute is claimed by some other peer. This can be achieved through active communication with other peers. This escalation process runs until the collection of peers is stable. The collection of peers is stable when no more attribute can be escalated. At the time a collection of peers is stable, communities have been formed as the set of peers that share the same sets of claimed attributes. Experiments show that peers stabilize after one level of indirection. Khambatti assumes automatic escalation for forming peer communities.
The attribute escalation algorithm facilitates the formation of the communities but is unconscious of its community memberships. After the attribute escalation process, a peer has the list of its claimed attributes and the subset of the claimed attributes that were escalated. If signatures are equal to the attributes that have been recently escalated, then the peer can be considered a member of the communities.

Definition (Kambatti et al. [24]) Let \( i \) be a node and \( C_i \) be a set that contains attributes claimed by \( i \). Consider a non-empty set \( N \) of nodes. Then the set resulting from the intersection of \( C_k \), for all \( k \in N \) is called a signature of the set \( N \).

In [24], link weights are used to determine the membership of a peer in a community. Link weight is the weight calculated for each claimed attribute of a peer \( V \) based on the number of links from \( V \) that can reach, after at most one indirection, other peers that claim the same attribute. Then a community discovery algorithm is proposed where link weights of claimed attributes are calculated, threshold values are assumed, and link weights are compared with the threshold values. The peer will assume membership in the community with claimed attribute set if the link weight of each claimed attribute is greater than a threshold value. The threshold value may be low or high, depending on the number of peers in the communities.

Vassileva [38] proposes an approach for supporting file and service sharing in research groups and groups of learners. She uses a multi-agent system for peer help called Community Gnutella (COMUTELLA) that motivates users to participate in the community in P2P systems. For getting help, a University of Saskatchewan student sends his query through his agent who finds other students who are currently on-line and have expertise in the area related to his query. She forms user communities based on a user’s
long-term interests, likelihood to search frequently in the same area, and common patterns of behavior. Vassileva designs user modeling to create user groups based on their common interests. The user model is composed of interests, resources (files or services), and relationships of the user. The users have to participate in a community which is a key to success in a P2P system and it is obviously crucial to motivate users in participating in communities. The author observed several levels of user cooperative participation which are: create service, allow service, facilitate search, allow communication, and perform uncooperative free riding. Free riders do not shoulder their fair share of the costs of their use of a resource, involvement in a project, etc. She assumes the users have long-term interests and are likely to search frequently in the same area at different times.

2.3.3 Community-based Search Techniques

Community-based search techniques have been considered in the context of P2P systems [12, 25, 26, 27, 37].

Kambatti et al. [25, 26, 27] consider a community-based search (CBS) technique that allows search operations based on peer content rather than only on filenames. Peer communities are based on the idea of "interest groups." Semantic analyzers can implicitly discover interest attributes, but the system design of Kambatti et al. obtains entire interests in an explicit manner. They propose a set of rules to be enforced for new peers to join the P2P systems. A search query has three parts: a) the identity of the peer creating the query, b) the actual query for an item, and c) a list of meta-information that describes the item. In the community-based search, a peer might use interest attributes as meta-information to a query. For processing the query, bloom filter summaries are used to
determine the appropriate communities. A bloom filter summary is a compact data structure that probabilistically represents the elements of a set. In the work of Khambatti et al., the bloom filter summary represents a compact summary of the attributes claimed by the peer members of a particular community and keeps the summary into a vector.

There are three approaches for selecting the first community peer where the processing of a query starts. Let $P_S$ be this peer. Then the three approaches are: a) $P_S$ is the closest seer (highly popular peers in a community) of a community whose signature $S$ contains at least one attribute from the meta-information list; b) $P_S$ is the seer in the community with $S$ matching the maximum meta-information attributes; and c) $P_S$ is the seer of a community whose signature $S$ matches the most important attributes from a weighted meta-information list of attributes. Khambatti et al. also provide a push-pull gossiping algorithm for information searching within P2P communities. The gossiping (push) phase improves the scalability of the algorithm. The efficiency of the pull phase depends exclusively on the efficiency of searches in the P2P system.

Gnasa et al. [12] present the architecture of Virtual Knowledge Communities (VKCs) which are built on Peer search memory (PeerSy), and describe the steps in creating representatives of VKCs from a set of $n$ peers. The peer search memory stores the search results locally in P2P networks to retrieve the web content. An Internet search engine, like Google, executes its general query-process, and activates the PeerSy database which sends out queries to the peer networks and matches with the representatives of different VKCs. If the query-process does not match any VKCs, then it finds out VKCs through a Seldom Asked Queries (SAQ) list. If no match is found, the query-link is inserted into the SAQ-list.
Sripanidkulchai et al. [37] propose interest-based shortcuts to link peers that share similar interests closer together. In this way, peers can benefit through direct cooperation. This technique is used on top of the Gnutella network to enhance the performance of P2P systems. The creation of shortcuts is based on the Gnutella topology, which depends on flooding queries to all peers. For the first few searches, the peer uses the underlying Gnutella topology. One of the return peers is selected from random and added to the shortcut lists. The subsequent queries go through the shortcuts lists first. If shortcut lists fails, it issues a lookup through the underlying Gnutella topology and repeats the process for adding new shortcuts. Each peer allocates a fixed-size amount of storage which limits to 10, and removes the low utility shortcut link when the list is full. Each shortcut is ordered by many metrics, such as success rate, path latency, load characteristics (query packets per peers process in the system), query scope (the fraction of peers in each query), and additional state kept in each node. The useful shortcut ranks at the top of the list, and an improving rank is determined by an increasing success rate. This technique uses exact matches for finding content locations in a peer. Peers send queries sequentially to each peer in the shortcut structure until the content is found. The shortcut structure is considered as a directed graph where vertices are peers and edges are shortcut relationships. In the shortcut structure, peers are grouped based on interests. Peers store performance history for all of their shortcuts due to the restriction of storage capacity.

2.4 Summary and Conclusions

This chapter presented methods for forming and discovering communities, as well as community-based search techniques which are directly related to our work. Next it
presented some approaches to web based communities, P2P search techniques, P2P information systems, peer-to-peer databases, query translation using mapping tables, and federated and multi-database systems.

Our work introduces a novel community-based search in peer database networks. Our technique is closely related to the work of Khambatti. For this reason, we mention the following shortcomings of his work. His community formation, discovery, and search techniques are based on text files. Though his work does not relate to peer databases, he has given concrete ideas regarding P2P community formation and discovery. He investigates randomly created communities, but does not clearly describe the creation of them. The community discovery algorithm takes several iterations to discover the membership of a community. In Khambatti’s community-based search technique, there are three approaches for selecting where to start the search. For the third approach, the criteria for determining the most important attributes as well as the ordering/sorting process of the meta-information of a query item are not clearly described. This community-based search technique is applicable to specific applications like digital libraries, where common interests are well-defined and understood by peer members. Sometimes communities will not form using the mentioned techniques due to a lack of matching common interests.

Our community formation, discovery, and search algorithms are based on peer databases instead of text files. We did not create communities randomly; rather communities are formed based on real commonly claimed attributes. Our algorithm reduces extra iterations for discovering communities. Our techniques extract meta-information easily from the condition parts of a given query. If a query has no condition
part then query considers all attributes of database as meta-information. We did not mention the format of interests of peers. Our solution overcomes most of the shortcomings above and provides an algorithm for community-based search in peer database networks.
Chapter 3

Querying Communities of Interest

3.1 Community-based Querying Method

The main ingredients of our community-based query processing approach are queries, communities, and peer databases. This section describes the methodology of community-based query with a running example, the pseudo-code of our algorithm, and the architecture of our community-based search. It also illustrates all the terms that are used in this chapter.

3.1.1 Basic Definitions

Before dealing with the community-based querying method, the following definitions should prove useful.

1. Peer-to-peer network: A peer-to-peer network is a network where there is no central server (computer) controlling the network or hierarchy among the computers. It relies on computing power at the edges (ends) of a connection rather than in the network itself.

2. Peer database network: A peer database network is a peer-to-peer network where each peer contains databases locally and shares these databases among themselves.

3. Flat file: A flat file is a code file that contains generally two columns, one for the codes, and the another one for the description of the codes. It is a database table.
Main file: A main file is a master file where all necessary information is kept. It contains data of some or all of the attributes as a code for security purposes. It is a main table of the database.

T-query: A T-query is a tabular form of data. It is used in a query translation algorithm to get answers from other peers. It translates a given query using join operation (OR) for each row.

ItemFile: ItemFile is used to store the description of attributes of a main file. More specifically, it provides the full name of each attribute of the main table of a database.

Database graph: A Database graph is a graphical and relational representation of different tables of a database.

CommunityBoard: A CommunityBoard is a data structure that stores community peers identity (e.g., peer name, signature of claimed attributes set).

Attribute escalation: This is a process in which attributes are promoted from a private attribute of a peer to a claimed attribute set to form communities.

Mapping rule: This is a rule that is imposed for data mapping in peer-to-peer data sharing systems.

Mapping Expression: The following is an inductive definition of mapping expression.

- Every mapping $\mu$ is a mapping expression
- If $\mu$ be a mapping expression then $\neg \mu$ is also a mapping expression
- If $\mu_1$ and $\mu_2$ be two mapping expressions then $\mu_1 \land \mu_2$ and $\mu_1 \lor \mu_2$ are also mapping expressions

A set of attributes to match a meta-information of a given query means the attributes of the conditional part or attributes of a given query is similar to a set of attributes.
Our solution has two steps:

1. Forming and discovering communities.
2. Translating a given query to be sent to the desired communities.

Step 1: Communities are discovered through database attributes/items. The attributes of peer databases are linked with ItemFile. Each peer claims some attributes from ItemFile then calculates the link weight of these claimed attributes. After that, it compares this link weight with a globally predetermined threshold value. Calculation of this threshold value is based on a number of observations. This is a "magic number" that is unscientifically assigned. If the link weight is greater than the threshold value then the peer belongs to a community with those attributes as signature. Each peer may belong to one or more communities. At the end of Step 1, every peer knows about communities to which it belongs and which attributes it claims.

Step 2: The next task is to translate the given query in order to search the desired communities. Each peer determines to which communities to send the query by checking attributes of the meta-information list of a given query with claimed attributes of a given peer in community. Meta-information is the conditional part of an SQL query. First, the querying peer searches for a peer where the processing of a given query starts. Let Ps be this peer. The querying peer searches Ps arbitrarily. Mujtaba [27] proposes three approaches for finding Ps, but our solution does not need those approaches because in peer databases we are using SQL queries whose results will be accurate and exact at all times. In the context of text files, results are not exact at all times. Partial results are also accepted. The given query is copied into a CommunityBoard. Periodically and asynchronously each neighboring peer visits the CommunityBoard, and if it can answer
the given query, then it sends results to the querying peer. The visiting neighbor peer also
copies the given query into its own CommunityBoard. This process repeats until all peers
of the community have been checked. The given peer uses mapping tables to translate the
query. If mapping tables linking the query peer with the target peer are empty, then the
peer uses direct peer databases for translating the query. Mapping tables between the set
of attributes are given in each peer. The given query is translated only for those peers that
belong to the desired community.

3.2 Motivating Example of Community-based Search

Consider ten peers $P_1, \ldots, P_{10}$ of movie communities as our example. Each peer
has personal and claimed attributes. Assume that the querying peer is $P_1$. Suppose that
peer sends an "Action related movie" query, which is denoted as $P_Q$. We assume that the
ID of the peer $P$ is $X$, and that the desired query is in SQL. Moreover, the meta-
information of the query $P_Q$ is "year - 2003", "Type - Action", and "CountryName -
America". Figure 3.7 illustrates the movie domain that we use as a running example.

3.2.1 Peer Schemas and Database Graph

A peer schema is the local database that mentions attributes of the main file and all flat
files. Each peer contains one or more databases. In our example, we are showing the
aforementioned movie schema. To be brief, we here describe only peers $P_1, P_7$, and $P_{10}$.

Peer $P_1$ Schema:

- Movie (MovieID, DirName, TypeID, CountryCode, AwardType, ActName, Year)
- MovieName (MovieID, MovieTitle)
- MovieType (TypeID, Type)
- Director (DirName, CountryCode, MovieID)
Country (CountryCode, CountryName, MovieID)
Award (AwardType, AwardName, MovieID)

Figure 3.1: Database Graph of Peer P1

A database graph (e.g. Figure 3.1) shows the relationship between the main file and flat files. More specifically, it shows join attributes between the main file and flat files.

Peer P1 Schema:

MovieDesc (MovID, Country, MovCategory, AwardCategory, ActorName, ActressName, MovieYear)
Movie (MovID, MovName)
Category (MovCategory, CategoryName)
Award (AwardCategory, AwardTitle)
Peer P_1 and P_7 contain the same attributes but with different names. From Figures 3.1 and 3.2, and the respective database schema, we see that P_1 has five flat files whereas P_7 has three flat files.

Peer P_{10} Schema:

Film (MovieName, MovieType, CountryOfMovie, MovieAward, ActorActress, YearOfMovie)

---

Figure 3.2: Database Graph of Peer P_7

Figure 3.3: Database Graph of Peer P_{10}
From Figure 3.3 and the P_{10} database schema, we see that P_{10} does not contain all the attributes that P_1 and P_7 contain. It is worth mentioning that P_{10} does not have any flat files. The remaining peers also contain similar movie schemas.

### 3.2.2 ItemFile of Schemas

An ItemFile stores data about data. The ItemFile of our example describes attributes and description of the main file of each peer. Each descriptive item is linked with one attribute of the main schema.

In order to recognize the claimed attributes in every peer, ItemFile is needed. An ItemFile resides in the front-end whereas the database is linked with the ItemFile in the back-end. Peers declare claimed attributes through ItemFile, but the processing (e.g. calculation of link weight, discover communities, etc.) is done by the related database.

For simplicity, ItemFile of peer P_1, P_7, and P_{10} are described here, but the rest of the peers contain each likewise an ItemFile with schema link.

Figure 3.4: ItemFile with Schema Link of Peer P_1.
Figure 3.4 represents the ItemFile of peer P₁ and links with database schema as well. The ItemFile of peer P₁ shows that the “Movie Name (N)” item is linked with the “MovieID” attribute of “Movie” database. The “Film Director (D)” item is linked with “DirName” attribute. Similarly, other items are linked with other attributes of the same database.

Figure 3.5: ItemFile with Schema Link of Peer P₇.

The ItemFile of peer P₇ in Figure 3.5 shows that the “Movie Name (N)” item is linked with the “MovID” attribute of “MovieDesc” database. The “Movie Type (T)” item is linked with “MovCategory” attribute. In the same way, other items are linked with other attributes of the same database.
Figure 3.6: ItemFile with Schema Link of Peer $P_{10}$.

Figure 3.6 illustrates the ItemFile of peer $P_{10}$ and shows its link with the related schema. The figure shows that "Movie Name (N)" item is linked with the "MovieName" attribute of "Film" database. The "Country of Movie (C)" item is linked with "CountryOfMovie" attribute. Likewise, other items are linked with other attributes of the same database.

### 3.2.3 Attribute Mappings

Different peers may contain the same attributes under different names. Through attribute mappings, schemas of two different peers match with their same attributes. In Table 3.1 and 3.2, only attribute mappings needed for our motivating example are shown. The motivating SQL example query Q is represented in Figure 3.7.
Table 3.1: Attribute Mapping between P₁ and P₇

<table>
<thead>
<tr>
<th>MovieID</th>
<th>MovID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>MovieYear</td>
</tr>
<tr>
<td>TypeID</td>
<td>MovCategory</td>
</tr>
<tr>
<td>CountryCode</td>
<td>Country</td>
</tr>
</tbody>
</table>

In Table 3.1, MovieID is mapped to the MovID attribute because these two attributes declare the same types of data of peer P₁ and P₇. Similarly, Year is mapped to MovieYear, TypeID is mapped to MovCategory, and CountryCode is mapped to Country.

Table 3.2: Attribute Mapping between P₇ and P₁₀

<table>
<thead>
<tr>
<th>MovID</th>
<th>MovieName</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieYear</td>
<td>YearOfMovie</td>
</tr>
<tr>
<td>MovCategory</td>
<td>MovieType</td>
</tr>
<tr>
<td>Country</td>
<td>CountryOfMovie</td>
</tr>
</tbody>
</table>

In Table 3.2, MovID, MovieYear, MovCategory, and Country of peer P₇ is mapped with the same attributes of peer P₁₀.

3.2.4 Mapping Tables

A mapping table associates values residing in different peers and provides a mechanism for sharing data. Mapping tables and mapping expressions are the basic tools for exchanging information between peers [2]. We use mapping tables for discovering communities in some cases, but they are mainly used for query translation of different peers. Both single and multiple mapping tables are usually used for query translation. In
this example, we use only a single mapping table. The creation and maintenance of mapping tables are done by a database administrator.

Table 3.3: Mapping Table between YearToMovieYear

<table>
<thead>
<tr>
<th>Year</th>
<th>MovieYear</th>
</tr>
</thead>
<tbody>
<tr>
<td>1982</td>
<td>82</td>
</tr>
<tr>
<td>1990</td>
<td>90</td>
</tr>
<tr>
<td>1999</td>
<td>99</td>
</tr>
<tr>
<td>2001</td>
<td>01</td>
</tr>
<tr>
<td>2002</td>
<td>02</td>
</tr>
<tr>
<td>2003</td>
<td>03</td>
</tr>
</tbody>
</table>

Table 3.3 shows the data value mapping of the “year” attribute between peer P₁ and P₇. In peer P₁, “year” is mentioned as four digits (e.g. 1982), whereas in peer P₇, the same attribute is declared as two digits (e.g. 82). Through this mapping table, it is easily recognized that year 1982 is equivalent to year 82. Now, Peer P₁ and P₇ can communicate and recognize the year field through mapping table 3.3.

Table 3.4: Mapping Table between MovieYearToYearOfMovie

<table>
<thead>
<tr>
<th>MovieYear</th>
<th>YearOfMovie</th>
</tr>
</thead>
<tbody>
<tr>
<td>82</td>
<td>1980</td>
</tr>
<tr>
<td>90</td>
<td>1990</td>
</tr>
<tr>
<td>99</td>
<td>1990</td>
</tr>
<tr>
<td>01</td>
<td>2000</td>
</tr>
<tr>
<td>02</td>
<td>2000</td>
</tr>
<tr>
<td>03</td>
<td>2000</td>
</tr>
</tbody>
</table>

Similarly, Table 3.4 indicates the data value mapping of the “year” attribute between peer P₇ and P₁₀. In peer P₇, “year” is defined as two digits (e.g. 99), but peer P₁₀ is defined as four digits (e.g. 1990’s) in terms of decades, not specific years.
Table 3.5: Mapping Table between TypeIDToMovieCategory

<table>
<thead>
<tr>
<th>Type</th>
<th>CategoryName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>War</td>
</tr>
<tr>
<td>Action</td>
<td>Fighting</td>
</tr>
<tr>
<td>Drama</td>
<td>Romantic</td>
</tr>
<tr>
<td>Drama</td>
<td>Love Story</td>
</tr>
<tr>
<td>Horror</td>
<td>Comedy</td>
</tr>
<tr>
<td>Horror</td>
<td>Futuristic</td>
</tr>
</tbody>
</table>

Table 3.5 shows the data value mapping of “movie category” between peer P1 and P7. In peer P1, action related movies are defined as “Action,” whereas in P7, the same type of movie is defined as “War” and “Fighting”. Similarly, drama movies are declared in peer P1 as “Drama,” whereas the same type of movie is declared in peer P7 as “Romantic” and “Love Story.”

Table 3.6: Mapping Table between CountryNameToCountry

<table>
<thead>
<tr>
<th>CountryName</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>America</td>
<td>USA</td>
</tr>
<tr>
<td>Canada</td>
<td>CA</td>
</tr>
<tr>
<td>England</td>
<td>UK</td>
</tr>
<tr>
<td>India</td>
<td>IND</td>
</tr>
</tbody>
</table>

Table 3.7: Mapping Table between CountryToCountryOfMovie

<table>
<thead>
<tr>
<th>Country</th>
<th>CountryOfMovie</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>North America</td>
</tr>
<tr>
<td>CA</td>
<td>North America</td>
</tr>
<tr>
<td>UK</td>
<td>Europe</td>
</tr>
<tr>
<td>IND</td>
<td>South Asia</td>
</tr>
</tbody>
</table>
In Table 3.6 and 3.7 the values of Country of peers P₁ and P₇, as well as those of peers P₇ and P₁₀ are shown. In peer P₇, country is mentioned in abbreviation (e.g. CA, IND), whereas in peer P₁₀, country is mentioned in continent (e.g. Europe).

3.2.5 Running Example

The movie community contains the following attributes, which are denoted as a letter. In the diagram of Figure 3.7, letters are used instead of full attributes names. The attributes are:

- Movie name = N
- Film Director = D
- Movie type = T
- Country of movie = C
- Movie award = A
- Actor/Actress = H
- Year of movie = Y

Each peer has a CommunityBoard, but only the CommunityBoard of the initial Ps (peer P₇) is shown in order to keep the diagram simple. In Figure 3.7, the SQL query, and the meta-information are also shown. Similarly, local database schemas, the database graph and the ItemFile of peer P₁ (querying peer), P₇ (initial Ps), and P₁₀ (neighbor peer of Ps) are shown in the corresponding Figure 3.1, Figure 3.4, Figure 3.2, Figure3.5, Figure 3.3, and Figure 3.6.
Figure 3.7: Diagram of Movie Community

We assume that communities are already formed using an attribute escalation method where attributes are claimed through the ItemFile of peer databases. Communities are discovered by computing the link weight and comparing the latter with the predetermined threshold values. We set a 60% threshold value in our implementation.

Communities vary with an increase or decrease in threshold value. The claimed attributes list of each peer and the peer identity of communities are created when communities are discovered. Now, the claimed attributes of each peer is checked against the meta-information list of the query to find the initial Ps. The user query is written in SQL and the meta-information is stored in an array which is extracted from the condition
parts of the SQL query. If there is no condition part, then all attributes are considered as meta-information. The Ps stores this SQL into its CommunityBoard, translates this query and sends the results to the querying peer. Other visiting peers (neighbour peers) check the CommunityBoard of Ps and copies this SQL into its own CommunityBoard. The visiting peer sends its result (if any) to the querying peer. All visiting peers of a community check their claimed attributes against the meta-information; if there is a match the query is translated. This process continues until the members of the community have been exhausted.

We use the following running example about movies. Assume that a user wants to retrieve all action related American movies shot in the year 2003 from peer P1. Then the Query “Q” should be:

Q: Select MovieTitle, Type, CountryName, year

   From Movie, MovieName, MovieType, Country

   Where Movie. MovieID= MovieName. MovieID

   AND Movie. TypeID= MovieType. TypeID


   AND Type= "Action"

   AND CountryName= "America"

   AND Year=2003

To retrieve information from the peer P1 database, the query should be as follows:

Q': Select MovName, CategoryName, Country, MovieYear

   From MovieDesc, Movie, Category

   Where MovieDesc. MovID= Movie. MovID
AND MovieDesc. MovCategory= Category. MovCategory
AND (CategoryName= "War" OR CategoryName= "Fighting")
AND Country= "USA"
AND MovieYear=03

To retrieve information from the peer P_{10} database, the query should be as follows:

Q'. Select MovieName, MovieType, CountryOfMovie, YearOfMovie
From Film
Where MovieType= "Action"
AND CountryOfMovie= "North America"
AND YearOfMovie=2000

To get information from the rest of the peers of movie communities, the original query Q should be translated to the vocabulary of the remote peers using the mappings that are in place.

3.2.6 Working Procedure of Our Example

In the context of our running movie community example, we are describing how each query is translated after searching the desired community and getting expected results from the rest of the community's peers. We assume that each peer in the network shares some attributes over the network. Each peer then defines mapping rules and the attribute-mapping table.

Peer P_{1} poses SQL query (P_{Q}) which has three constraints. From this SQL query, we get the following meta-information: year (2003), movie type ("Action"), and country ("America"). Peer P_{1} has these claimed attributes, so the user gets results from P_{1}. Then query P_{Q} looks for Ps arbitrarily in the same community. As we have claimed attributes
of each peer and the identities of all peers of each community, so \( P_Q \) travels within its community. We assume \( P_Q \) checks claimed attributes of peer \( P_7 \) with the meta-information of the given query. The claimed attributes of peer \( P_7 \) matches with the meta-information of a given query, so we can consider peer \( P_7 \) as \( P_s \). Peer \( P_7 \) then stores this query \( P_Q \) in the CommunityBoard and starts a translation process to get results. If a mapping table is found it applies; otherwise it uses mapping rules and attributes of the mapping of the local peer database. The database graph is used to find relations and necessary join operations. We did not show any mapping rules because our example query does not need mapping rules. For query \( P_Q \) we need attribute mapping and a mapping table.

For translating the given query, \( P_Q \) performs the following tasks:

First, \( P_Q \) is converted to its corresponding T-Query \( (Q_T) \) which is:

<table>
<thead>
<tr>
<th>MovieTitle</th>
<th>Type</th>
<th>CountryName</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>&quot;Action&quot;</td>
<td>&quot;America&quot;</td>
<td>2003</td>
</tr>
</tbody>
</table>

Second, computes T-Query \( (Q'_T) \) by joining \( Q_T \) with mapping Table 3.5 of peer \( P_7 \):

<table>
<thead>
<tr>
<th>MovName</th>
<th>CategoryName</th>
<th>Country</th>
<th>MovieYear</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>&quot;War&quot;</td>
<td>&quot;USA&quot;</td>
<td>03</td>
</tr>
<tr>
<td>M2</td>
<td>&quot;Fighting&quot;</td>
<td>&quot;USA&quot;</td>
<td>03</td>
</tr>
</tbody>
</table>

Finally, the \( Q'_T \) translates the query using a join operation (OR) for each row. We notice that the database graph of Figure 3.2 subsumes attributes MovName, CategoryName, Country, and MovieYear. From the graph of Figure 3.2 we see that we need join relations for MovieDesc, Movie, category, and Country. The complete translated query is as follows:
\[ Q: \text{Select MovName, CategoryName, Country, MovieYear} \]
\[ \text{From MovieDesc, Movie, Category} \]
\[ \text{Where MovieDesc. MovID= Movie. MovID} \]
\[ \text{AND MovieDesc. MovCategory= Category. MovCategory} \]
\[ \text{AND (CategoryName= "War" OR CategoryName= "Fighting")} \]
\[ \text{AND Country= "USA"} \]
\[ \text{AND MovieYear=03} \]

Then the visiting peer \( P_{10} \) (neighbor of \( P_7 \) or \( P_s \)) of the community checks the query into the CommunityBoard of \( P_7 \). If there is a match then a translation of the given query is started; otherwise \( P_{10} \) copies this query (\( P_0 \)) into its own CommunityBoard. The visiting peer of \( P_{10} \) (suppose \( P_6 \)) checks the CommunityBoard of \( P_{10} \). If there is a match then the query is translated and results are sent to the querying peer; otherwise it sends nothing, but copies the query into its own CommunityBoard. This procedure repeats for all of members in the community.

### 3.3 Algorithm for Community-based Search

The complete pseudo-code of our algorithm is given below:

\[ \text{Membership Form} \]

\[ /* \text{Executes at each peer simultaneously */} \]

\[ \text{Begin} \]

\[ \text{Define ItemFile for attributes of peer database} \]

\[ \text{Declare the claimed items set from ItemFile} \]

\[ \text{Receive the claimed items set from neighbourhood ItemFile} \]
Find intersection of personal item set with received claimed item set

If received claimed item is mapped to personal item, but not in personal set

Add claimed item in intersection set

End-If

Foreach item of intersection set

If item is not present in declared claimed items

Add item to declared claimed items

End-If

End-For

End

The differences between Khambatti and our membership formation algorithms are that we use peer databases instead of text files. We also use mapping tables to get claimed items in intersection sets.

Membership Discovery

/* Executes at each peer simultaneously */

Begin

Foreach claimed items

Compute link weight of each item of every peer

/* Calculation of link weight */

Link weight=0

Foreach peer P

Foreach neighbour P₁ of P

If neighbour P₁ contains claimed items
**Link weight** = Link weight + 1

End-If

**Foreach** neighbour of P_1

**If** neighbour contains claimed items

**Link weight** = Link weight + 1

End-If

End-For

End-For

End-For

/* Calculation of link weights of each peer is done by analyzing the claimed item sets that it receives from neighbouring peers. */

/* End of link weight calculation */

**If** link weight > T  /* T= Threshold value */

Consider as a *community item*

End-If

End-For

/* [Number of Community Signature= 2^i-1; ‘i’ is number of community items] */

For all subsets S of *community items*

Peer is a member of community with signature S

Set a Vector for community [v]

/* Keep community information in a vector array */

End-For
Store community peers ID in CommunityBoard

Refresh communities within a time interval

End

Our algorithm first calculates the link weight of all claimed attributes and compares the link weight with the threshold value. Khambatti’s algorithm computes the link weight of all claimed attributes and then checks the link weights for the subset of claimed attributes with threshold value. Our algorithm does not check the subset of claimed attributes. For this reason, our algorithm reduces reiteration.

Community-based search using Query Translation

/* Translate the given query in searching desired community */

Begin

Set identifier P_Q for the peer creating the query

Mark the actual query as being the original query

Set meta-information [i] for the query item

/* meta-information is defined from condition part of SQL query or all attributes*/

Check Community peers /* Check the desired community in each peer */

If claimed items of vector for community [v] match with meta-information [i]

Ps is determined

Convert query Q to its corresponding T-Query Q_T=T /* Q is an SQL query */

/* Ps is the first searching community peer where processing of query starts. */

If mapping table is not empty

Computes T-query Q_T=Π_(i=1)^n T (▷ m) /* m= mapping table, ▷=attribute*/
Else

Computes \( Q'_\tau = \prod U' (T) \) from local peer database

End-If

Obtain the query \( Q' \) from \( Q'_\tau \) using disjunction for each row of attributes

/* \( Q' \) is a SQL query */

Copy the Query into its CommunityBoard and Send result to \( P_Q \)

Neighbour peer of community checks CommunityBoard

If claimed items of Neighbour peer match with meta-information

Translate the query and Send result to \( P_Q \)

Copy the Query into its own CommunityBoard

Else

Copy the Query into its own CommunityBoard

End-If

Else

Search for \( P_S \)

End-If

End

3.4 Discussion of Our Algorithm

Our algorithm has three parts: (1) forming communities using an attribute escalation method; (2) discovering communities comparing link weight with the threshold value; (3) translating the given query using T-query in the appropriate communities, which is called community-based search. The community-based search
procedure calls T-Query for translating the given query. The aforementioned community-based search can be done using an alternative approach.

The alternative approach for community-based query is simple. After discovering communities, every peer knows to which community it belongs. The given query checks its meta-information with the claimed attributes of the CommunityBoard. It searches for community peers and translates the given query for those peers in the querying peer. Then the querying peer sends the translated query directly to those community peers in order to get results.

The alternative approach is the following:

Pseudo-code:

*Community-based search using Query Translation (Alternative approach)*

/* Translate the given query in searching desired community */

**Begin**

**Identify** querying peer ID

**Mark** the actual query as being the original query

**Identify** meta-information [i] for the query item

/* meta-information is defined from condition part of SQL query or all attributes*/

**Check** Community peers /* Check the desired community in each peer */

**If** claimed items of vector for community [v] match with meta-information [i]

**Search** for community peers, whose claimed items match with meta-information

**Convert** query Q to its corresponding T-Query Q\_T /* Q is an SQL query */

**If** mapping table is not empty
Computes T-query $Q'_T = \prod_{U'/T} (T \Join m)$ /* $m$ = mapping table, $u'$ = attribute */

Else

Computes $Q'_T = \prod_{U'/T} (T)$ from local peer database

End-If

Translate the query $Q'$ from $Q'_T$ using disjunction for each row of attributes

/* $Q'$ is an SQL query */

Send translated query into those peers for getting result in querying peer

End-If

End

The alternative approach is given for comparing with our community-based search algorithm which is mentioned in section 3.3. Our community-based search algorithm processes the given query in a community peer and proceeds for the next peer of a community. But, in the alternative approach, the algorithm does not process and proceeds for the next community peers. As each peer knows about his communities peers, the alternative approach can process the given query from the querying peer.

3.5 System Architecture of Community-based Search

Figure 3.8 represents a generic architecture for our community-based peer DBMS with its main functionalities. This architecture extends the Hyperion DBMS architecture with a community-based search component. All the modules of our community-based search architecture are described in the following.
Figure 3.8: Architecture for a Community-based Search of a P2P System

This architecture consists of four main components: a interface (GUI), a P2P layer, a Community Discovery (CD) layer, and a DBMS. The GUI is the interface for posing queries (local or global) and specifying whether these are to be executed only locally or remotely.

The P2P layer has four sub-components: a Rule Manager (RM), a Update Manager (UM), a Query Manager (QM), and an Acquaintance Manager (AM). Through an AM, the P2P layer allows a peer database to establish or abolish an acquaintance (semi-) automatically at run time, thereby inducing a logical peer-to-peer network. The
AM uses mapping tables as a constraint on the data exchange between peer databases to automatically check the consistency of a set of mapping constraints and to infer new ones. The QM and the UM are responsible for queries and update propagation. They manage domain relations, coordination formulas, coordination rules, acquaintances, and interest groups. The RM enforces stabilization and consistency policies between peers. The Peer DB contains local databases, mapping tables, and mapping expressions that are used in data exchange with other peers.

The CD layer has four sub-components: an Attribute Escalator/Item Matcher (AE/IM), a Community Formator (CF), a Link weight Calculator (LC), and a Community Discovery Rule Manager (CDRM). The CD layer is connected with the QM and AM for propagating the given query into a desired community. The AE/IM escalates attributes/items by intersecting personal items with claimed items. The CF is responsible for forming communities. The LC calculates link weight of each of the claimed attributes. Finally, the CDRM discovers communities by comparing link weight with predetermined threshold values.
4.1 Motivation

Efficient discovery of communities reduces the communication cost of community-based search. If communities are discovered in an optimized way, then query results among communities will be faster. The term "optimization" usually presumes the system retains the same functionality. However, a significant improvement in performance can often be achieved by solving only the actual problem and removing irrelevant functionality. In computing, optimization is the process of modifying a system to improve its efficiency. Our goal is to send the query after forming and discovering communities, or to discover communities ‘on the fly’ at the time of propagating the given query. For this reason, we are optimizing the community discovery procedure. The focus of this chapter is to discuss Khambatti’s community discovery technique with a competent and optimized approach, and to elaborate shortly the idea of discovering communities ‘on the fly’. This chapter also explains experimental results and discusses our prototype implementation.

4.2 Khambatti’s Community Discovery Technique

Khambatti’s community discovery algorithm is presented here to identify the difference between his community discovery algorithm and an optimized community discovery algorithm. His algorithm computes the link weight of all claimed attributes. It then checks the link weight of each claimed attribute for the subset of claimed attributes.
If all the attributes of each subset are greater than the pre-determined threshold value, then the peer is a community member of that subset. Khambatti’s community discovery algorithm is as follows:

**Membership Discovery (peer)**

/* execute at each peer simultaneously */

**Foreach** claimed attributes

**Compute** link weight

**End-for**

**Foreach** subset S of claimed attributes set

**Foreach** attribute in S

**If** link weight > T then

Check next attribute

**Else**

**Break-for** /* try another subset */

**End-for**

**If** no break-for was executed

Peer is member of community with signature S

**End-for**

### 4.3 Optimizing Khambatti’s Community Discovery Technique

Our algorithm is the optimized form of Khambatti’s community discovery algorithm. The optimized algorithm first calculates the link weight of all claimed attributes and compares the link weight with the threshold value. If the link weight is greater than the threshold value, then that attribute is considered as community items. In
this way, the algorithm finds the community items. After that, the algorithm finds the
subset of these community items and all these subsets are the members of the community.
The optimized algorithm is as follows:

*Membership Discovery*

/* Executes at each peer simultaneously */

**Foreach** claimed items

  **Compute** link weight of each attribute in every peer

  **If** link weight > T /* T= Threshold value */

     Consider as a *community item*

  **End-If**

**End-For**

/* [Number of Community Signature= 2^{i-1}; 'i' is number of community items] */

**For all** subsets S of *community items*

  Peer is a member of community with signature S

  **Set a Vector** for community [v]

  /* Keep community information in a vector array */

**End-For**

4.4 **Benefit of Modified Technique**

The modified community discovery technique reduces the comparing cost. As a
result, it takes fewer iterations and processing is faster than Khambatti’s algorithm.
Suppose a peer has 10 claimed attributes. In this case, the peer will have $2^{10}$ (1024)
subsets. If the link weight of 5 claimed attributes of that peer is less than the threshold
value, then our modified algorithm takes $2^5$ (32) iterations, instead of $2^{10}$ (1024)
iterations. But Khambatti’s algorithm takes $2^{10}$ (1024) iterations, which is redundant. Our algorithm is faster than Khambatti’s community discovery algorithm.

4.5 Discovering Communities ‘on the fly’

4.5.1 Communities ‘on the fly’

Communities can be discovered in a local machine (each peer), in a distributed way, or ‘on the fly’. This section introduces how communities can be discovered ‘on the fly’. Discovering community ‘on the fly’ means that when a peer poses a query then it propagates to the acquainted peers and discovers communities at runtime.

We assume that each peer calculates the link weight of each attribute of every acquainted peer or any initiator peer calculates the link weight and sends this calculated link weight file to the group where it belongs. Link weight is the key factor in discovering communities. Then we have to set a threshold value. The threshold value compared with the link weights generates the signature of the community. In this way, communities can be discovered.

If the discovered communities match with the received query criteria, then the query is translated to retrieve results from this peer.

4.5.2 Algorithm for Discovering Community ‘on the fly’

The pseudo-code is as follows:

Begin

Calculate/Receive calculated link weight of each claimed attributes from initiator peer

Set a threshold value
If receive message from neighbor peer

    Compare link weights of attributes mentioned in the message with

    Threshold value (T)

    For each claimed attributes whose link weight > T

        Peer belongs to community with Signature S

    End-For

    If Query Criteria match discovered community

        Translate the given query for this peer

    End-If

End-If

End

4.5.3 Discussion

The key factor for discovering community ‘on the fly’, which is computing the
link weights, is the same as the regular discovery of communities in a local peer. The
only difference is that communities are discovered in run time when a peer poses a query.
The initiator peer sends the posed query to its acquaintees then, upon receipt, each
acquainted peer starts discovering to which communities. After discovering communities
at runtime, the algorithm translates the given query for communities peers. The advantage
of this mechanism is that the peer does not need to store community information. As a
result, every peer optimizes storage capacity. The main disadvantage is to discover
communities in every execution time and the algorithm needs more memory space. The
query completion time for this case is slower than our regular community-based search.
4.6 Implementation

We have implemented our community-based search algorithm which is mentioned in section 3.3. We have done experiments on our implemented algorithm. Section 4.7 shows our experimental results. We have not implemented our discovering community ‘on the fly’ algorithm which is mentioned in section 4.5.

Figure 4.9: Screen-shot of the GUI of Declared Claimed Items

Our system is written in Java SDK 1.4.2 on top of JXTA, a P2P programming framework that alleviates the burden of writing huge source code. Its implementation consists of approximately 4000 lines of code. The prototype system runs on the Microsoft Windows environment. Figure 4.9 and 4.10 show screenshots of the query interface of our system.
Figure 4.9 shows the main interface for claiming attributes and managing claim acceptances and rejections. When a peer declares claimed attribute, then one message is generated for all acquainted peers. The acquainted peers then accept or reject their interest by pressing the Accept/Reject button. When a user chooses the “Accept” button, the information is sent to the corresponding text files of all acquainted peers.

![Diagram of query interface](image)

Figure 4.10: Screen-shot of Query Generation, Translation, and Community Discovery

Figure 4.10 shows the query interface which serves for generating queries, translating them, and discovering communities. It shows the community formation and link weight calculation and the community discovery menus as well. After generating the query, the peer translates it for community peers and sends the translated query to those
peers for getting query results. The user has to select schema, attributes, and predicates for generating queries. The given query uses mapping tables for translating the generated query.

4.7 Experiments

Our system uses MySQL 4.1.9 as our DBMS for the local DB layer. This system is built on top of JXTA. In our experiments, we found the performance of Java SDK 1.4.2 satisfactory. We provide a user interface for declaring claimed attributes/items, discovering communities, and posing a query to the peer database networks. The algorithm for community-based search is the combination of Khambatti’s community formation and discovery technique, and the query translation method of Kementssietsidis using mapping table concepts. We develop an algorithm keeping the above ideas with supplementary modifications. We are mainly concerned with measuring the correctness and efficiency of discovering communities. To measure the efficiency and effectiveness of our algorithm, we did some experiments. We created a real network with 10 peers on 4 different machines using the JXTA package. We observed that the algorithms work well.

To measure the efficiency and effectiveness of our algorithm, we did some experiments. Because of the unreliability of JXTA pipes at the time of the experiments, and motivated by the desire to create a significiation amount of peers, we decided to create two networks of 100 and 300 peers, respectively, all on one single machine. We experimented by setting threshold value at 40%, 50% and 70%. We get fewer communities with increasing threshold values. As we mentioned earlier (see section 3.1.1) threshold values are determined based on observations. Based on the fact that at
around 60%, sufficiently many communities are formed, we decided to set the link threshold to 60%. We collected data for evaluating the following:

- Community refreshment within different time intervals;
- Effect of query propagation for community changes;
- Percentage of community discovered for different sized peer networks.

### 4.7.1 Community Refreshment within Different Time Intervals

To do this measurement, we first had to find out the number of communities that were discovered given different time intervals. This information is then used to calculate the percentages of community changes. The X-axis represents time in minutes; the Y-axis denotes the corresponding percentage of community changes.

![Graph showing community refreshment over time](image)

**Figure 4.1: Community Refreshment by Changing Peers/Nodes.**

Initially, we ran our simulation with 100 peer networks. For plotting percentage community changes, we gathered two sets of data: one is by changing peers (join or leave) from networks, the other is by varying attributes from peers within different time intervals. In our model, there are 4 text files and these are: network file, personal attribute file, claimed attribute file, and mapping table file. When a user accepts claimed
attributes, the corresponding claimed attributes and peer name are stored in a text file named claimed.txt, similarly when peers join the network, the peer name and link are stored in a text file named network.txt. In our experiment, joining or leaving from networks means to add or delete peer name from the network.txt text file. We manually change entries in network.txt file. In our networks, each peer contains a maximum of 12 personal attributes, and a peer was explicitly placed into only one group.

![Graph](image)

**Figure 4.2: Community Refreshment by Changing Attributes/Items.**

The graphs shown in Figure 4.1 and 4.2 are the results of our community refreshment within different time intervals. The curves of Figure 4.1 and 4.2 are in different shapes. The percentage of communities change depends on how many peers change their status after community refreshment. The large time interval for community refreshment is more efficient because the experiment shows that a few users change their position (leave or join) over a small time interval. For attribute changes, the percentage of community changes is high and always almost over 80%, but for peer/node changes, the percentage of community changes is not so high and varies often. Our observation determines that a peer changes its attributes more frequently than to join or leave networks.
4.7.2 Effect of Query Propagation for Community Changes

For this evaluation, we send a query; change the percentage of communities, and record query completion time. In our 100 peer network, we placed approximately 300 mapping tables.

![Graph](image1)

Figure 4.3: Effect of Community Changes on the Completion Time a Given Query:
SELECT MovieID, Year FROM movie WHERE MovieID="M110" OR Year="00"

![Graph](image2)

Figure 4.4: Effect of Community Changes on the Completion Time a Given Query:
SELECT MovieID, Year FROM movie WHERE MovieID="M110" AND Year="00"

The graphs shown in Figures 4.3 and 4.4 are the results of our experiment. The X-axis indicates percentage of community changes and Y-axis represents query completion time in seconds. We send different queries by varying percentage communities and plot separately.
The curve of Figure 4.3 goes down, then up, and then down whereas Figure 4.4 goes up, then down, and then up. In the first few communities change of Figure 4.3, the response of peers decreases for disjunctive query. Again response of peers increases. The response of peers depends on attribute matching of a given query with its claimed attributes. Similarly, in the first communities change of Figure 4.4, the response of peers increases for conjunctive query. Again response of peers decreases. Due to the increase and decrease of peers response, the curves of Figure 4.3 and 4.4 go up and down. We have done our experiments on fixed sizes of network and our declared claimed attributes, but it may vary at a time in real P2P environment.

The completion time for a specific query does not depend on the size of the query. Query completion time depends on how many peers reply after changing communities. We did not send complex queries since that was not our goal. Our target was to observe the query completion time for given percentages of community changes. We sent two SQL queries where the condition part of the first one was a conjunction (AND), and the second one was a disjunction (OR). Experiments indicate that simple disjunctive queries take approximately double the time that conjunctive queries do.

### 4.7.3 Percentage of Communities Discovered for Different Sizes of Peer Networks

We ran our algorithm in different sized peer networks and recorded the number of communities discovered. We started to run our prototype with 50 peers, increasing 50 peers every time, and taking the measurement up to 200 peers.
Figure 4.5: Effect of Communities Discovered for Different Sizes of Networks

Figure 4.5 depicts percentage of community discovered for different sized networks. The X-axis represents number of peers/nodes and the Y-axis represents percentage of communities discovered. We observe that the percentage of communities discovered remains the same all the time. The number of communities for each peer should be \((2^i-1)\), where ‘i’ is the number of claimed attributes whose link weight is greater than predetermined threshold value. We verified that the number of discovered communities is correct. We also verified the calculated link weight for validating our discovered communities. The number of discovered communities remains constant for different sized networks and discovers 100% of communities.

4.7.4 Complexity of Our Algorithm

A community-based search algorithm is our main objective. This algorithm has three major parts: community formation, community discovery, and query translation using T-query. In order to measure the overall complexity, we first discuss the
complexity of related procedures and finally we determine the complexity of our community-based search algorithm.

The development of fast and efficient algorithms is very important for computers. Generally, to estimate the complexity of an algorithm, we need to determine how much time and space is required to implement it. This requires determining the number of steps that a program will take to solve a problem. To count steps, it is important to consider the order of growth of a problem. The complexity of an algorithm can be determined theoretically and empirically. We measure the complexity of our algorithm empirically, concentrating on how long it takes to perform our desired experimental operation.

A community can be discovered in each peer, or in a distributed way, or ‘on the fly’. We implemented the community discovery technique in each peer simultaneously. The running time depends on the programming language that is used for implementation and the machine that the program is run on. As our main input elements are attribute and node/peer, we input different sizes of attributes and nodes/peers separately or collectively. We then observe the results of community formation and link weight calculation time, community discovery time, community search of a given query & query translation time for determining the complexity of the algorithm.

We analyze the complexity of our algorithm with the following experiments. We started using 12 attributes and increased up to 50. The required time for the same operation remains the same for nodes or attributes change. For simplicity, only the operations for node changes are shown.
Figure 4.6: Community Formation and Link Weight Calculation

Figure 4.6 illustrates community formation and link weight calculation time in seconds for different sized networks. We started with 50 nodes and increase by 50 nodes every time. We gather data 6 times until 300 nodes to observe community formation and link weight calculation time. From the above graph, we see that community formation and link weight calculation time increases gradually in terms of node increases. This algorithm does not take long to perform the task. We see that time increases slightly (milliseconds) for node changes.

Figure 4.7: Community Discovery Times

Figure 4.7 depicts the community discovery time in terms of different sized networks. The same number of nodes and iterations we used for community formation
and link weight calculation experiment are used here. We set the link threshold to 60%. From the above graph we see time increases progressively for node changes. Community discovery method takes longer than link weight calculation time. The increasing time for community discovery is much more than to form community for the same network changes. The reason is that for discovering communities the algorithm compares link weight with threshold values every time whereas for community formation and link weight calculation it only calculates values, there is no comparison.

![Graph](image)

Figure 4.8: Overall Query Processing Time for a Typical Simple Query

Figure 4.8 shows the community search of a given query and query translation time for different sized networks. This is our main module. This module is the community-based search module. Figure 4.8 shows that time increases almost proportionally in terms of network changes. The increasing time for network changes is in between link weight calculation and community discovery module. The required increasing time is in seconds but is not high.

In every case, the function for time complexity is different, but time increases gradually in terms of node increases. The coefficient slightly varies in all cases. The
community discovery module takes longer in comparison with other module (e.g. link weight calculation, query translation, etc.). But we do not need to discover communities every time. We need to refresh communities in a certain interval for accurate performance. However, the community-based search module is not slower. Based on the experiments, we can observe that our algorithm takes gradual time increments in terms of node increments.
Chapter 5

Conclusion and Future Work

5.1 Conclusion

P2P systems offer several advantages in simplicity of use, robustness, and scalability. In this thesis we address the issue of forming and discovering communities, and pose a query to get response from appropriate communities. We analyze some techniques for discovering communities in a text files and then present an algorithm.

Our prototype consists of five components: an attribute declaration mechanism, link weight calculation module, community discovery mechanism, a query generator tool, and a query translation scheme for community peers. The attribute declaration mechanism declares claimed attributes and propagates to the group. The acquainted peer accepts/rejects its interest regarding these claimed attributes. If it accepts, a message is sent to all peers each of which stores this information in its claimed text file. The link weight calculation module calculates link weight and stores this result in a text file. The Community discovery mechanism discovers communities by taking information from claimed and link weight text files. A query generator tool generates the query based on predicate selection, and a query translation scheme translates the given query for every community peer through mapping tables.

Our experimental results indicate that community discovering remains accurate in different network sizes. Experiments show that we need more memory swap space if we increase the number of claimed attributes. In that case, the community sorting period also
increases. Experiments also indicate that the threshold value has a strong impact on discovering communities. So, we have to determine the threshold value in a careful way, otherwise we will not reach our expected communities.

Users can identify their peer names as numeric or alpha numeric also. We develop our prototype on top of JXTA, which supports P2P platform environment. An approach for discovering community ‘on the fly’ is introduced.

5.2 Future Work

For future work we plan to implement the algorithm for discovering communities ‘on the fly’. We plan to collect real data and use them to experiment on our prototype. Currently, our system accepts only simple queries (e.g. AND, OR), but complex queries could be incorporated in our algorithm as well.

We further plan to discover communities in a distributed way. Automatic mapping table generation is necessary for query translation purposes, which is not our objective, but in some cases it is required to discover an accurate community. There is also potential to extend our prototype so that it can run in a real environment.

Based on observations, it is recommended that if there are a large number of claimed attributes in the network, then any single peer should work on a portion of claimed attributes for discovering communities and other peers should work on some portion of attributes; similarly the next peer should work on some of the attributes to discover communities until the end of claimed attributes. After discovering communities for all claimed attributes, then all discovered communities in a file should be merged and sent to all peers in the community.
BIBLIOGRAPHY


[33] Piazza peer data management system (pdms), (Online) Available at: http://data.cs.washington.edu/p2p/piazza


### APPENDIX A

#### SAMPLE OUTPUTS

1. **Link weight of each attribute for every peer**

    (For simplicity & to save space, 10 out of 100 peer link weights are shown)

<table>
<thead>
<tr>
<th>PeerName</th>
<th>ClaimedAttribute</th>
<th>ClaimedDBField</th>
<th>LinkWeight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Type_Of_Movie</td>
<td>TypeID</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>Name_Of_Movie</td>
<td>MovieID</td>
<td>61</td>
</tr>
<tr>
<td>0</td>
<td>Country_Movie</td>
<td>CountryCode</td>
<td>59</td>
</tr>
<tr>
<td>0</td>
<td>Cost_Of_Movie</td>
<td>MovCost</td>
<td>58</td>
</tr>
<tr>
<td>0</td>
<td>Main_Singer</td>
<td>Singer</td>
<td>43</td>
</tr>
<tr>
<td>0</td>
<td>Film_Director</td>
<td>DirName</td>
<td>59</td>
</tr>
<tr>
<td>0</td>
<td>Theatre_Name</td>
<td>Theatre</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>Year_Of_Movie</td>
<td>Year</td>
<td>68</td>
</tr>
<tr>
<td>0</td>
<td>Movie_Award</td>
<td>AwardType</td>
<td>60</td>
</tr>
<tr>
<td>0</td>
<td>Best_Selling</td>
<td>Selling</td>
<td>58</td>
</tr>
<tr>
<td>0</td>
<td>Movie_Producer</td>
<td>Producer</td>
<td>55</td>
</tr>
<tr>
<td>1</td>
<td>Theatre_Name</td>
<td>Theatre</td>
<td>67</td>
</tr>
<tr>
<td>1</td>
<td>Year_Of_Movie</td>
<td>Year</td>
<td>68</td>
</tr>
<tr>
<td>1</td>
<td>Type_Of_Movie</td>
<td>TypeID</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>Name_Of_Movie</td>
<td>MovieID</td>
<td>61</td>
</tr>
<tr>
<td>1</td>
<td>Country_Movie</td>
<td>CountryCode</td>
<td>57</td>
</tr>
<tr>
<td>1</td>
<td>Cost_Of_Movie</td>
<td>MovCost</td>
<td>58</td>
</tr>
<tr>
<td>1</td>
<td>Main_Singer</td>
<td>Singer</td>
<td>44</td>
</tr>
<tr>
<td>1</td>
<td>Film_Director</td>
<td>DirName</td>
<td>58</td>
</tr>
<tr>
<td>1</td>
<td>Best_Selling</td>
<td>Selling</td>
<td>59</td>
</tr>
<tr>
<td>1</td>
<td>Movie_Producer</td>
<td>Producer</td>
<td>57</td>
</tr>
<tr>
<td>2</td>
<td>Name_Of_Movie</td>
<td>MovieID</td>
<td>43</td>
</tr>
<tr>
<td>2</td>
<td>Type_Of_Movie</td>
<td>TypeID</td>
<td>47</td>
</tr>
<tr>
<td>2</td>
<td>Main_Singer</td>
<td>Singer</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>Film_Director</td>
<td>DirName</td>
<td>44</td>
</tr>
<tr>
<td>2</td>
<td>Movie_Award</td>
<td>AwardType</td>
<td>44</td>
</tr>
<tr>
<td>2</td>
<td>Best_Selling</td>
<td>Selling</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>Movie_Producer</td>
<td>Producer</td>
<td>41</td>
</tr>
<tr>
<td>3</td>
<td>Name_Of_Movie</td>
<td>MovieID</td>
<td>33</td>
</tr>
<tr>
<td>3</td>
<td>Year_Of_Movie</td>
<td>Year</td>
<td>38</td>
</tr>
<tr>
<td>3</td>
<td>Main_Singer</td>
<td>Singer</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>Film_Director</td>
<td>DirName</td>
<td>33</td>
</tr>
<tr>
<td>3</td>
<td>Theatre_Name</td>
<td>Theatre</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>Best_Selling</td>
<td>Selling</td>
<td>58</td>
</tr>
<tr>
<td>4</td>
<td>Movie_Award</td>
<td>AwardType</td>
<td>63</td>
</tr>
<tr>
<td>4</td>
<td>Type_Of_Movie</td>
<td>TypeID</td>
<td>67</td>
</tr>
<tr>
<td>4</td>
<td>Name_Of_Movie</td>
<td>MovieID</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>Country_Movie</td>
<td>CountryCode</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>Actor_Actress</td>
<td>ActName</td>
<td>51</td>
</tr>
<tr>
<td>5</td>
<td>Best_Selling</td>
<td>Selling</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>Name_Of_Movie</td>
<td>MovieID</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>Theatre_Name</td>
<td>Theatre</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>Actor_Actress</td>
<td>ActName</td>
<td>23</td>
</tr>
<tr>
<td>Movie_Award</td>
<td>AwardType</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>Year_Of_Movie</td>
<td>Year</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Film_Director</td>
<td>DirName</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>Main_Singer</td>
<td>Singer</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>MovieProducer</td>
<td>Producer</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Actor_Actress</td>
<td>ActName</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>Name_Of_Movie</td>
<td>MovieID</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>Film_Director</td>
<td>DirName</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Year_Of_Movie</td>
<td>Year</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>MovieProducer</td>
<td>Producer</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>Theatre_Name</td>
<td>Theatre</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>Cost_Of_Movie</td>
<td>MovCost</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Movie_Award</td>
<td>AwardType</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Country_Movie</td>
<td>CountryCode</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>Type_Of_Movie</td>
<td>TypeID</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>Best_Selling</td>
<td>Selling</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>Actor_Actress</td>
<td>ActName</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>Theatre_Name</td>
<td>Theatre</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>Year_Of_Movie</td>
<td>Year</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>Type_Of_Movie</td>
<td>TypeID</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>Name_Of_Movie</td>
<td>MovieID</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>Country_Movie</td>
<td>CountryCode</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>Cost_Of_Movie</td>
<td>MovCost</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Film_Director</td>
<td>DirName</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>Best_Selling</td>
<td>Selling</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>MovieProducer</td>
<td>Producer</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>Movie_Award</td>
<td>AwardType</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>Theatre_Name</td>
<td>Theatre</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>Type_Of_Movie</td>
<td>TypeID</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>Name_Of_Movie</td>
<td>MovieID</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>Country_Movie</td>
<td>CountryCode</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>Year_Of_Movie</td>
<td>Year</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>Movie_Award</td>
<td>AwardType</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>Best_Selling</td>
<td>Selling</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>MovieProducer</td>
<td>Producer</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>Type_Of_Movie</td>
<td>TypeID</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Cost_Of_Movie</td>
<td>MovCost</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>MovieProducer</td>
<td>Producer</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>Best_Selling</td>
<td>Selling</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>Actor_Actress</td>
<td>ActName</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>Main_Singer</td>
<td>Singer</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Theatre_Name</td>
<td>Theatre</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Name_Of_Movie</td>
<td>MovieID</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Year_Of_Movie</td>
<td>Year</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Name_Of_Movie</td>
<td>MovieID</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>Country_Movie</td>
<td>CountryCode</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>Year_Of_Movie</td>
<td>Year</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>Movie_Award</td>
<td>AwardType</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Film_Director</td>
<td>DirName</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Type_Of_Movie</td>
<td>TypeID</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>Actor_Actress</td>
<td>ActName</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>Cost_Of_Movie</td>
<td>MovCost</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Main_Singer</td>
<td>Singer</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Theatre_Name</td>
<td>Theatre</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>MovieProducer</td>
<td>Producer</td>
<td>39</td>
<td></td>
</tr>
</tbody>
</table>
2. **Signature of community with members (peers)**

(For simplicity & to save space, not all communities are shown)

Signature of Community with member peers:

<table>
<thead>
<tr>
<th>MovieID = Name Of Movie, DirName = Film_Director, TypeID = Type_Of_Movie, CountryCode = Country_Movie, AwardType = Movie_Award, ActName = Actor_Actress, Year = Year_Of_Movie, Theatre = Theatre_Name, Selling = Best_Selling, MovCost = Cost_Of_Movie, Singer = Main_Singer, Producer = MovieProducer</th>
<th>Community Signature</th>
<th>Peer Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[TypeID]</td>
<td>0,1,2,4,6,7,8,9,10,12,15,17,19,21,22,23,24,25,26,28,29,31,32,33,34,35,38,39,41,42,43,44,45,46,50,51,52,53,54,60,64,73,75,76,77,86,89,91,93,96,98,99</td>
<td>[MovieID]</td>
</tr>
<tr>
<td>Type ID</td>
<td>Country Code</td>
<td>Singer</td>
</tr>
<tr>
<td>---------</td>
<td>--------------</td>
<td>--------</td>
</tr>
<tr>
<td>0,1</td>
<td>21,34,35,50</td>
<td>98</td>
</tr>
<tr>
<td>0,1</td>
<td>7,17,21,22,28,34,46</td>
<td>99</td>
</tr>
<tr>
<td>0,1</td>
<td>7,8,10,17,22,25,34,39</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,8,17,21,22,24,35,38,45,46,99</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,8,22,28,34,50</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,17,22,25,28,29,34,45,99</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>8,10,17,22,25,28,29,34,45,99</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,8,22,24,34,35,52,54</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,19,21,22,24,28,38,42,60</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>8,22,24,34,38,42,44,52,76</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,9,22,24,38,42,44,52,76</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,9,22,24,38,42,44,52,76</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,9,9,19,22,24,28,34,44,52,60,76</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,9,9,19,22,24,28,29,34,42,44,45,54,60</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>2,15,21,24,38</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>9,24,34,38,52,75</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>9,24,38,52,75,89</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>2,13,34,35,82,52,75</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>2,13,54,34,50,52,89</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>2,13,24,34,75</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,17,22,24,34,38,42,46,91</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,12,17,22,24,38,42,77,98</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,12,17,22,24,34,38,42,60,99</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>1,7,9,15,19,22,24,28,34,60,91</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,9,12,17,22,24,28,34,60,99</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,9,12,17,22,24,28,34,60,99</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,9,12,17,22,24,28,34,60,99</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,15,22,24,28,34,44,52,76,89,91</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,12,17,22,24,34,38,46,52,75,76</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,12,17,22,24,28,34,44,75</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,8,12,17,22,34,38,44,52,75,76</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,8,15,22,24,28,34,44,52,76,89,93</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,8,10,17,22,24,25,28,34,44,75,93</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>2,7,8,22,34,44,52,76</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>2,7,8,17,22,34,44,75</td>
<td></td>
</tr>
<tr>
<td>0,1,2</td>
<td>7,8,19,22,24,25,28,34,44,52,60,93</td>
<td></td>
</tr>
<tr>
<td>0,1,7</td>
<td>18,21,22,28,29,31,35,39,71,91,96</td>
<td></td>
</tr>
<tr>
<td>0,1,14</td>
<td>21,30,31,35,50,71</td>
<td></td>
</tr>
<tr>
<td>0,1,7</td>
<td>17,18,21,22,28,91,99</td>
<td></td>
</tr>
<tr>
<td>0,1,7</td>
<td>17,18,22,38,39,91</td>
<td></td>
</tr>
<tr>
<td>0,1,7</td>
<td>8,10,17,18,22,39,71</td>
<td></td>
</tr>
<tr>
<td>0,7</td>
<td>8,17,18,21,22,31,35,38,71,99</td>
<td></td>
</tr>
<tr>
<td>0,1,6</td>
<td>7,8,22,28,50,91</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,8,10,17,22,28,29,71,99</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>21,31,35,40,52,54,71</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,18,19,21,22,28,60,91</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,18,22,44,49,52,54,91</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,18,22,44,49,52,53,71</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>6,7,18,21,22,31,35,44,52,60,61,71,97</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>6,7,19,22,28,44,52,60,91</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>7,19,22,28,29,44,53,54,60,71</td>
<td></td>
</tr>
<tr>
<td>0,1,2</td>
<td>3,15,21,91</td>
<td></td>
</tr>
<tr>
<td>0,3</td>
<td>9,52,75,91</td>
<td></td>
</tr>
<tr>
<td>0,9</td>
<td>52,71,75</td>
<td></td>
</tr>
<tr>
<td>0,2</td>
<td>21,35,52,71,75</td>
<td></td>
</tr>
<tr>
<td>0,1,2</td>
<td>15,50,52,91</td>
<td></td>
</tr>
<tr>
<td>0,1,2</td>
<td>5,71,75,91</td>
<td></td>
</tr>
<tr>
<td>MovieCost, Theatre, Selling</td>
<td>${0, 7, 22, 24, 34, 44, 52, 76, 91}</td>
<td></td>
</tr>
<tr>
<td>MovieCost, Theatre, Producer</td>
<td>${0, 7, 22, 24, 34, 44, 57}</td>
<td></td>
</tr>
<tr>
<td>MovieCost, Year, AwardType</td>
<td>${0, 7, 18, 22, 38, 44, 52, 64, 71, 76, 94}</td>
<td></td>
</tr>
<tr>
<td>MovieCost, Year, Selling</td>
<td>${0, 7, 22, 24, 44, 52, 76, 93}</td>
<td></td>
</tr>
<tr>
<td>MovieCost, Year, Producer</td>
<td>${0, 7, 22, 24, 44, 53, 57, 71, 93}</td>
<td></td>
</tr>
<tr>
<td>MovieCost, AwardType, Selling</td>
<td>${0, 6, 7, 22, 34, 37, 44, 52, 76}</td>
<td></td>
</tr>
<tr>
<td>MovieCost, AwardType, Producer</td>
<td>${0, 7, 22, 34, 44, 57, 71}</td>
<td></td>
</tr>
<tr>
<td>MovieCost, Selling, Producer</td>
<td>${0, 1, 7, 19, 22, 24, 28, 34, 44, 60, 93}</td>
<td></td>
</tr>
<tr>
<td>Singer, DirName, Theatre</td>
<td>${0, 3, 18, 24, 38, 39, 91}</td>
<td></td>
</tr>
<tr>
<td>Singer, DirName, Year</td>
<td>${0, 18, 24, 38, 39, 77}</td>
<td></td>
</tr>
<tr>
<td>Singer, DirName, AwardType</td>
<td>${0, 2, 18, 21, 37, 38}</td>
<td></td>
</tr>
<tr>
<td>Singer, DirName, Selling</td>
<td>${0, 1, 2, 15, 19, 24, 37, 91}</td>
<td></td>
</tr>
<tr>
<td>Singer, DirName, Producer</td>
<td>${0, 1, 2, 19, 24}</td>
<td></td>
</tr>
<tr>
<td>Singer, Theatre, Year</td>
<td>${0, 9, 18, 24, 25, 39, 52, 75, 77}</td>
<td></td>
</tr>
<tr>
<td>Singer, Theatre, AwardType</td>
<td>${0, 18, 34, 38, 52, 75}</td>
<td></td>
</tr>
<tr>
<td>Singer, Theatre, Selling</td>
<td>${0, 24, 34, 52, 91}</td>
<td></td>
</tr>
<tr>
<td>Singer, Theatre, Producer</td>
<td>${0, 24, 25, 34, 75, 77}</td>
<td></td>
</tr>
<tr>
<td>Singer, Year, AwardType</td>
<td>${0, 18, 38, 52, 71, 75, 94}</td>
<td></td>
</tr>
<tr>
<td>Singer, Year, Selling</td>
<td>${0, 24, 52, 89}</td>
<td></td>
</tr>
<tr>
<td>Singer, Year, Producer</td>
<td>${0, 24, 25, 71, 75}</td>
<td></td>
</tr>
<tr>
<td>Singer, AwardType, Selling</td>
<td>${0, 2, 34, 37, 52}</td>
<td></td>
</tr>
<tr>
<td>Singer, AwardType, Producer</td>
<td>${0, 2, 34, 71, 75}</td>
<td></td>
</tr>
<tr>
<td>Singer, Selling, Producer</td>
<td>${0, 1, 2, 19, 24, 25, 34, 77}</td>
<td></td>
</tr>
<tr>
<td>DirName, Theatre, Year</td>
<td>${0, 7, 17, 18, 22, 24, 39, 42, 44, 52, 75, 89}</td>
<td></td>
</tr>
<tr>
<td>DirName, Theatre, AwardType</td>
<td>${0, 6, 7, 17, 18, 22, 31, 34, 38, 44, 46, 52, 75}</td>
<td></td>
</tr>
<tr>
<td>DirName, Theatre, Selling</td>
<td>${0, 6, 7, 22, 24, 34, 44, 52, 89, 91}</td>
<td></td>
</tr>
<tr>
<td>DirName, Theatre, Producer</td>
<td>${0, 7, 17, 22, 24, 34, 44, 75}</td>
<td></td>
</tr>
<tr>
<td>DirName, Year, AwardType</td>
<td>${0, 6, 7, 12, 17, 18, 22, 31, 38, 44, 52, 75, 94, 96}</td>
<td></td>
</tr>
<tr>
<td>DirName, Year, Selling</td>
<td>${0, 6, 7, 22, 24, 44, 52, 89, 93}</td>
<td></td>
</tr>
<tr>
<td>DirName, Year, Producer</td>
<td>${0, 6, 7, 10, 17, 22, 24, 44, 75, 93, 96}</td>
<td></td>
</tr>
<tr>
<td>DirName, AwardType, Selling</td>
<td>${0, 2, 6, 7, 22, 33, 34, 37, 44, 52}</td>
<td></td>
</tr>
<tr>
<td>DirName, AwardType, Producer</td>
<td>${0, 2, 7, 17, 22, 33, 34, 44, 75}</td>
<td></td>
</tr>
<tr>
<td>DirName, Selling, Producer</td>
<td>${0, 1, 2, 14, 19, 22, 24, 28, 34, 44, 54, 60, 93}</td>
<td></td>
</tr>
<tr>
<td>Theatre, Year, AwardType</td>
<td>${0, 7, 8, 12, 17, 18, 21, 22, 35, 44, 45, 52, 61, 64, 71, 75, 76, 94, 99}</td>
<td></td>
</tr>
<tr>
<td>Theatre, Year, Selling</td>
<td>${0, 1, 7, 8, 22, 24, 28, 44, 52, 76, 89}</td>
<td></td>
</tr>
<tr>
<td>Theatre, Year, Producer</td>
<td>${0, 1, 5, 7, 8, 17, 22, 24, 25, 28, 44, 45, 57, 71, 75, 99}</td>
<td></td>
</tr>
<tr>
<td>Theatre, AwardType, Selling</td>
<td>${0, 6, 7, 8, 22, 34, 37, 44, 52, 76}</td>
<td></td>
</tr>
<tr>
<td>Theatre, AwardType, Producer</td>
<td>${0, 5, 7, 8, 17, 22, 34, 44, 71, 75}</td>
<td></td>
</tr>
<tr>
<td>Theatre, Selling, Producer</td>
<td>${0, 1, 7, 8, 22, 24, 28, 34, 44}</td>
<td></td>
</tr>
<tr>
<td>Year, AwardType, Selling</td>
<td>${0, 6, 7, 8, 22, 34, 44, 52, 76}</td>
<td></td>
</tr>
<tr>
<td>Year, AwardType, Producer</td>
<td>${0, 7, 8, 17, 22, 34, 44, 71, 75}</td>
<td></td>
</tr>
<tr>
<td>Year, Selling, Producer</td>
<td>${0, 1, 7, 8, 22, 24, 28, 34, 44, 93}</td>
<td></td>
</tr>
<tr>
<td>AwardType, Selling, Producer</td>
<td>${0, 2, 7, 8, 22, 34, 44, 54, 77}</td>
<td></td>
</tr>
<tr>
<td>TypeID, MovieID, CountryCode, MovCost</td>
<td>${0, 1, 7, 21, 22, 28, 29, 35}</td>
<td></td>
</tr>
<tr>
<td>TypeID, MovieID, CountryCode, Singer</td>
<td>${0, 1, 21, 35, 50}</td>
<td></td>
</tr>
<tr>
<td>TypeID, MovieID, CountryCode, DirName</td>
<td>${0, 1, 7, 17, 21, 22, 28, 99}</td>
<td></td>
</tr>
<tr>
<td>TypeID, MovieID, CountryCode, Theatre</td>
<td>${0, 7, 17, 22, 38}</td>
<td></td>
</tr>
<tr>
<td>TypeID, MovieID, CountryCode, Year</td>
<td>${0, 7, 8, 17, 22}</td>
<td></td>
</tr>
<tr>
<td>TypeID, MovieID, CountryCode, AwardType</td>
<td>${0, 7, 8, 17, 21, 22, 35, 38, 99}</td>
<td></td>
</tr>
<tr>
<td>TypeID, MovieID, CountryCode, Selling</td>
<td>${0, 1, 7, 8, 22, 28, 50}</td>
<td></td>
</tr>
<tr>
<td>TypeID, MovieID, CountryCode, Producer</td>
<td>${0, 1, 7, 8, 17, 22, 28, 59, 99}</td>
<td></td>
</tr>
<tr>
<td>TypeID, MovieID, MovCost, Singer</td>
<td>${0, 1, 21, 35, 52, 54}</td>
<td></td>
</tr>
<tr>
<td>TypeID, MovieID, MovCost, DirName</td>
<td>${0, 1, 7, 19, 21, 22, 28, 60}</td>
<td></td>
</tr>
<tr>
<td>TypeID, MovieID, MovCost, Theatre</td>
<td>${0, 7, 22, 44, 52, 54}</td>
<td></td>
</tr>
</tbody>
</table>
| AgentName, TypeID, MovieID, CountryCode, MovCost, DirName, Theatre, Year, AwardType, Selling, Producer | $\{7\}
3. The given query responses from a community

(For simplicity & to save space, a few responses from communities are shown)

The given Query is:
-----------------------------------------------
SELECT MovieID, Year FROM movie WHERE MovieID="M110" OR Year="00"
-----------------------------------------------

The given query will hit in the following peers of Community:

{ 0,1,2,3,4,5,6,7,8,9,10,12,14,15,16,17,18,19,20,21,22,27,28,29,30,31,32,33,35,38,39,40,42,44,47,49,50,
  52,53,54,55,57,60,61,62,65,70,71,75,77,81,82,85,90,91,93,96,97,99
  11,24,25,34,45,56,63,64,66,69,76,79,
  80,83,86,89,94,98
  }

Response from Communities:
-----------------------------------------------

[MovieID] -----> {0,1,2,3,4,5,6,7,8,9,10,12,14,15,16,17,18,19,20,21,22,27,28,29,30,31,32,33,35,38,39,40,42,44,47,49,50,
  52,53,54,55,57,60,61,62,65,70,71,75,77,81,82,85,90,91,93,96,97,99}
[Year] -----> {0,1,3,5,6,7,8,9,10,11,12,14,15,16,17,18,20,21,22,24,25,27,28,29,30,31,34,35,38,39,42,44,45,47,49,
  52,53,55,56,57,60,61,62,63,64,65,66,69,71,75,76,77,79,80,81,82,83,85,86,89,90,93,94,96,98,99}
[TypeID, MovieID] -----> {0,1,4,7,8,9,12,15,17,19,21,22,28,29,32,33,35,38,42,44,45,50,52,54,60,75,77,81,93,99}
[TypeID, Year] -----> {0,7,8,9,12,15,17,22,24,25,28,34,38,39,42,44,52,75,76,77,89,93,98}
[MovieID, CountryCode] -----> {0,1,4,6,7,8,10,12,14,17,18,21,22,28,29,30,31,35,38,39,53,71,91,96,99}
[MovieID, MovCost] -----> {0,1,6,7,10,18,19,21,22,28,29,30,31,35,39,44,49,52,53,54,60,61,71,91,96,97,99}
[MovieID, Singer] -----> {0,1,2,3,5,9,10,14,15,21,31,35,40,50,52,54,71,75,91}
[MovieID, DirName] -----> {0,1,2,3,5,6,7,9,10,12,15,16,17,18,19,21,22,28,40,42,50,57,77,90,91,99}
[MovieID, Theatre] -----> {0,3,5,6,7,9,10,12,17,18,22,30,31,38,39,42,44,49,52,54,57,75,91}
[MovieID, Year] -----> {0,3,5,6,7,8,9,10,12,14,17,18,20,22,30,31,39,42,44,47,49,52,53,55,57,71,75,77,85,90,93}
[MovieID, AwardType] -----> {0,2,5,6,7,8,10,12,17,18,21,22,31,35,38,40,44,52,53,55,60,61,71,75,81,96,97,99}
[MovieID, Selling] -----> {0,1,2,6,7,8,12,15,19,22,28,29,30,40,44,50,52,53,91,93}
[MovieID, Producer] -----> {0,1,2,5,6,7,8,10,17,19,22,28,29,30,42,44,53,54,57,60,71,75,91,93,99}
[CountryCode, Year] -----> {0,7,8,10,11,17,18,22,25,34,39,42,49,52,57,60,71,90}
[MovCost, Year] -----> {0,7,9,14,18,22,24,38,42,44,49,52,53,57,64,71,76,80,83,90,93,94,98}
[Singer, Year] -----> {0,9,18,24,25,30,38,39,52,71,75,77,89,94}
[DirName, Year] -----> {0,6,7,11,12,14,17,18,22,24,31,38,39,42,44,52,60,65,75,77,89,90,93,94,96,98}
[Theatre, Year] -----> {0,1,5,7,8,9,12,17,18,21,22,24,25,28,30,35,39,42,44,45,47,49,52,57,61,64,71,75,76,77,79,89,94,98,99}
[Year, AwardType] -----> {0,6,7,8,10,12,17,18,21,22,31,34,35,38,44,45,49,52,60,61,64,71,75,76,81,94,96,99}
[Year, Selling] -----> {0,1,6,7,8,15,22,24,28,29,30,34,44,52,76,86,89,93}
[Year, Producer] -----> {0,1,5,6,7,8,10,17,22,24,25,28,29,30,34,44,45,47,53,57,60,71,75,93,96,99}
[TypeID, MovieID, CountryCode] -----> {0,1,4,7,8,12,17,21,22,28,29,35,38,99}
[TypeID, MovieID, MovCost] -----> {0,1,7,9,21,22,28,29,35,44,52,54,60,99}
[TypeID, MovieID, Singer] -----> {0,1,9,15,21,35,50,52,54,75}
[TypeID, MovieID, DirName] -----> {0,7,9,12,17,22,38,42,44,52,54,75}
[TypeID, MovieID, Theatre] -----> {0,7,8,9,12,17,22,42,44,52,75,77,93}
4. Query translation for community peers of a given query

Partial translated Query for a given Query:

<table>
<thead>
<tr>
<th>Peer name</th>
<th>Translated query</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM cinema WHERE (MovieName=Titanic OR MovieName=Bob the Builder OR MovieName=Lord of the Ring) OR (MovieYear=2000 OR MovieYear=2001 OR MovieYear=2002 OR MovieYear=2003 OR MovieYear=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Terminator I)</td>
<td></td>
</tr>
<tr>
<td>FROM movie WHERE (MovieName=MacGyver OR MovieName=Alice in Wonderland OR MovieName=Harry Potter)</td>
<td></td>
</tr>
<tr>
<td>FROM filmdesc WHERE (FilmID=F003) OR (FilmYear=01 OR FilmYear=02 OR FilmYear=04 OR FilmYear=95)</td>
<td></td>
</tr>
<tr>
<td>FROM cinema WHERE (CinemaName=Six Million $ma OR CinemaName=Queen Elizabeth) OR (CinemaYear=2000 OR CinemaYear=2001)</td>
<td></td>
</tr>
<tr>
<td>FROM moviedesc WHERE (MovID=Tita OR MovID=Home OR MovID=Lord) OR (MovieYear=2000 OR MovieYear=2001 OR MovieYear=2002 OR MovieYear=2003 OR MovieYear=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM cinema WHERE (MovieName=Titanic OR MovieName=X Files OR MovieName=Dreamland)</td>
<td></td>
</tr>
<tr>
<td>FROM art WHERE (ArtName=Harry Potter OR ArtName=McDonald OR ArtName=Landlord)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM movie WHERE (MovieName=Airforce II OR MovieName=Bob the Builder)</td>
<td></td>
</tr>
<tr>
<td>FROM movie WHERE (MovieName=Airforce II OR MovieName=Bob the Builder)</td>
<td></td>
</tr>
<tr>
<td>FROM movie WHERE (MovieName=Airforce II OR MovieName=Bob the Builder)</td>
<td></td>
</tr>
<tr>
<td>FROM movie WHERE (MovieName=Airforce II OR MovieName=Bob the Builder)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM movie WHERE (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
<tr>
<td>FROM film WHERE (MovieName=Airforce II OR MovieName=Bob the Builder) OR (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)</td>
<td></td>
</tr>
</tbody>
</table>
FROM movie WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM cinema WHERE (MovieName=Airforce II)
FROM cinema WHERE (MovieName=Airforce II)
FROM art WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM cinema WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM art WHERE (MovieName=Airforce II)
FROM art WHERE (MovieName=Airforce II)
FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM movie WHERE (MovieName=Airforce II)
FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM movie WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM movie WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM movie WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM film WHERE (MovieName=Airforce II)
FROM movie WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM film WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM cinema WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM movie WHERE (MovieName=Airforce II) OR (YearOfMovie=2000 OR YearOfMovie=2004)
FROM film WHERE (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
FROM art WHERE (YearOfMovie=2000 OR YearOfMovie=2003 OR YearOfMovie=2004)
FROM film WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
FROM art WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
FROM art WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
FROM cinema WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
FROM art WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
FROM film WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
FROM film WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
FROM cinema WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
FROM film WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
FROM art WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
FROM cinema WHERE (YearOfMovie=2000 OR YearOfMovie=2004)
FROM cinema WHERE (YearOfMovie=2000 OR YearOfMovie=2004)