Towards a Comprehensive Web Service Recommendation Framework

by

Abdullah Abdullah

A Thesis presented to
The University of Guelph

In partial fulfilment of the requirements
for the degree of

Doctor of Philosophy

in

Computer Science

Guelph, Ontario, Canada
©Abdullah Abdullah, April, 2017
Web services (WS) nowadays are considered a consolidated reality of the modern Web with remarkable, increasing influence on everyday computing tasks. Following Service-Oriented Architecture (SOA) paradigm, corporations are increasingly offering their services within and between organizations either on intranets or the cloud. The aim of this work is to advance the academic efforts in assisting end users and corporations to benefit from Web service technology by facilitating the recommendation and integration of Web services into composite services.

In this thesis, we propose a recommendation framework that is capable of not only recommending an individual Web service but also a composite one when no service available to fulfill the user request. The framework is realized into two main parts. A recommendation model for individual WS is proposed where the QoS profile is considered as an implicit rating scheme. The model utilizes the Jaccard coefficient in several variants to create two Unipartite similarity-based graphs. By integrating them with the original user-service rating graph, a richer recommendation model is constructed. Using the Top-K Random Walk algorithm, a final set of recommendations is delivered to end user. The model proves its well-behaviour in terms of sparsity tolerance and recommendation accuracy. To minimize the complexity, a thresholding technique is proposed in which the Random Walk is
better guided using a reduced subset of users based on their Jaccard similarities. Furthermore, the applicability of the model as a generic recommendation model is also examined using an ordinary rating domain.

The second component is a service composition model in which AI-based planning using Agent technology is adopted to dynamically and flexibly construct composite service workflow. In this model, a distributed service dependency model based on AND/OR graph structure is decomposed and distributed among individual members of the Agent community. The agents are equipped with a well-defined internal reasoning mechanism based on agents’ knowledge. Using a communication protocol, the agents actively collaborate to find a cost-effective executable workflow according to end user request. Finally, feasibility and effectiveness demonstration of all components of the proposed framework, using publicly available datasets, a recommendation library, and a multi-agent platform is verified.
To my Parents and Family...
Acknowledgements

First and foremost, I would like to express my appreciation and gratitude to my advisor, Dr. Xining Li, who offered valuable support, understanding, and encouragement. His alenthusiasm inspired me greatly to work and I will forever be grateful to him.

I am sincerely and heartily grateful to the members of my advisory committee: Dr. Stefano Gregori and Dr. Qusay Mahmoud, who provided me with feedback throughout this long journey.

I would also like to thank my external examiner Dr. Jianguo Lu and Dr. Fangju Wang who also provided valuable suggestions on my thesis.

Last but not least, I would like to express my deepest gratitude to my immediate family for their exceptional love and support, my mother Khawla Ahmed, my father Asaad Abdullah, my lovely wife Noor Mohammed, and my kids, Omar, Fatimah, Hudhaifa, and Hemzah, as well as my extended family, who showered me with their constant encouragement and enduring patience...
Contents

Acknowledgements iv

Contents v

List of Figures ix

List of Tables xi

1 Introduction 1

1.1 Motivation ........................................ 1
1.2 Research Problem .................................. 3
1.3 Thesis Assumptions ................................ 6
1.4 Contributions ...................................... 7
1.5 Thesis Outline ..................................... 9

2 Background 10

2.1 Introduction to Recommender Systems ............ 11
2.2 Recommendation Techniques ...................... 11
2.2.1 Content-based Methods .......................... 12
2.2.2 Collaborative Filtering Methods ................. 15
2.2.2.1 Memory-based CF Methods .................... 16
2.2.2.2 Model-based CF Methods ...................... 18
2.2.3 Hybrid Methods .................................. 20
2.2.4 Advantages and Disadvantages of CF Approaches .... 20
2.3 Advanced Recommendation Techniques ............ 23
2.3.1 Dimensionality reduction methods ............... 23
2.3.2 Graph-based Methods ......................... 23
2.3.2.1 Random walk similarity ....................... 24
## 3 Literature Review

3.1 Web Service Recommendation ........................................ 28
3.2 Prior Work on Web Service Recommendation .................. 29
  3.2.1 Text-based Approaches ........................................... 29
  3.2.2 QoS-based Approaches .......................................... 32
  3.2.3 Semantic-based Approaches ................................... 35
  3.2.4 Usage-based Approaches ....................................... 38
3.3 Graph-based Recommendation ....................................... 38
3.4 Web Service Composition ............................................ 41
  3.4.1 AI Planning Composition ....................................... 41
  3.4.2 Prior Works on AI Planning Composition ....................... 43
  3.4.3 Sample Research Projects ..................................... 49
    3.4.3.1 Connected Living ......................................... 49
    3.4.3.2 MSG EUREF ................................................ 50
    3.4.3.3 IMA .......................................................... 51
    3.4.3.4 EMD .......................................................... 53
3.5 Summary and discussion .............................................. 54

## 4 Random Walk based Recommendation Model for Web Services

4.1 Introduction ............................................................ 57
4.2 Motivation ............................................................. 58
4.3 Contributions .......................................................... 59
4.4 WS recommendation model ............................................. 60
  4.4.1 User-Service Bipartite Graph .................................. 62
4.5 Similarity Graphs ....................................................... 63
  4.5.1 User-Based Unipartite Graph ................................... 64
  4.5.2 Service-Based Unipartite Graph ................................ 64
  4.5.3 Jaccard similarity for recommendation ......................... 64
4.6 Integrated-Model QoS-based Graph (IMQG) ......................... 69
4.7 Recommending Web Services using IMQG ........................ 71
  4.7.1 Generic Top-K Random Walk algorithm .......................... 71
  4.7.2 RW-based Recommender Algorithm for IMQG ..................... 72
4.8 Running Example ....................................................... 74
4.9 Evaluation and Experiments ......................................... 81
  4.9.1 Dataset .......................................................... 82
  4.9.2 Testing Methodology ............................................ 83
## 4.9.3 Recommendation Accuracy

- 4.9.3 Recommendation Accuracy ................................................. 86

## 4.9.4 Impact of Experimental Parameters

- 4.9.4.1 Top-K ............................................................... 90
- 4.9.4.2 Given Number ......................................................... 94
- 4.9.4.3 Training Matrix Density ............................................. 99

## 4.10 Summary ................................................................. 102

## 5 Enhanced Similarity based model for web service recommendation

- 5.1 Introduction ............................................................... 105
- 5.2 Contributions ............................................................. 105
- 5.3 Enhanced Recommendation Model ..................................... 106
- 5.4 Recommender Algorithm of Reduced IMQG ......................... 109
- 5.5 Evaluation of the Reduced IMQG ....................................... 111
  - 5.5.1 Recommendation Accuracy ........................................... 111
    - 5.5.1.1 Domain Reduction Ratio (DRR) .......................... 111
    - 5.5.1.2 Impact of Top-K ............................................ 113
    - 5.5.1.3 Impact of Jaccard Coefficient .......................... 114
  - 5.5.2 Computation Time .................................................. 116
    - 5.5.2.1 Given Number ............................................... 117
    - 5.5.2.2 Density ..................................................... 118
    - 5.5.2.3 Top-K ..................................................... 120
- 5.6 Generality of the Proposed Model .................................... 121
  - 5.6.1 Evaluation Environment ......................................... 121
  - 5.6.2 Evaluating The Proposed Model .................................. 123
  - 5.6.3 Evaluating The Reduced Model .................................. 125
  - 5.6.4 Comparison With Other Recommendation Approaches ........... 129
- 5.7 Summary ................................................................. 132

## 6 Agent-Based Model to Web Service Composition

- 6.1 Introduction ............................................................... 133
- 6.2 Agent and Service Technologies ....................................... 134
- 6.3 The Contributions .......................................................... 135
- 6.4 Basic Definitions .......................................................... 137
- 6.5 WSC Scenarios ............................................................ 139
- 6.6 Modelling of Agent Knowledge ........................................ 143
  - 6.6.1 AND/OR graph .................................................. 143
  - 6.6.2 AND/OR graph and Web Services ............................ 144
  - 6.6.3 Service Dependency AND/OR Graph ....................... 145
List of Figures

3.1 System Architecture [31] ................................. 30
3.2 WSExpress System Architecture [31] ......................... 32
3.3 The Architecture of the DAML-S Matching Engine [42] ........ 36
3.4 WSPlan architecture [56] .................................. 42
3.5 Xplan architecture [57] ..................................... 43
3.6 SHOP2 based system architecture [58] ....................... 44
3.7 Architectures of PORSCE II and VLEPPO [60] ............... 45
3.8 Extended AI Planner [63] .................................. 46
3.9 Geospatial Web Services composition system [65] .......... 48
3.10 Life cycle of agent-based web service composition [67] ..... 49
3.11 IMA platform is based on the distributed multi-agent framework JIAC, (ima.dai-labor.de) ......................... 52

4.1 WS Recommendation Model Construction ...................... 61
4.2 User-Service Bipartite Graph Example .......................... 63
4.3 The Creation of Transition Matrix $A_{n,n}$ Based on IMQG Model .......... 70
4.4 Comparison of Top-K Response Time ........................ 91
4.5 Comparison of Top-K Response Time ........................ 92
4.6 Comparison of Top-K Failure Rate .......................... 93
4.7 Comparison of Top-K Failure Rate .......................... 94
4.8 Comparison of Given Number ............................... 96
4.9 Comparison of Given Number ............................... 98
4.10 Comparison of Training Matrix Density ..................... 100
4.11 Comparison of Training Matrix Density ..................... 101

5.1 Recommendation Model Reduction .......................... 108
5.2 Domain Reduction Ratio vs. Recall .......................... 112
5.3 Recall-Optimized-SST .................................... 114
5.4 Precision-Optimized-SST .................................. 115
5.5 Comparison of SST vs. Recall-Optimized-Jaccard(s) .......... 116
5.6 Comparison of SST vs. Precision-Optimized-Jaccard(s) ....... 117
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.7</td>
<td>Comparison of Time vs. Given Number</td>
<td>118</td>
</tr>
<tr>
<td>5.8</td>
<td>Comparison of Time vs. Density</td>
<td>119</td>
</tr>
<tr>
<td>5.9</td>
<td>Comparison of Time vs. TopK</td>
<td>120</td>
</tr>
<tr>
<td>5.10</td>
<td>The Proposed Model Using FilmTrust</td>
<td>126</td>
</tr>
<tr>
<td>5.11</td>
<td>The Reduced Model Using FilmTrust</td>
<td>127</td>
</tr>
<tr>
<td>5.12</td>
<td>Comparisons with the Reduced Model (FilmTrust)</td>
<td>131</td>
</tr>
<tr>
<td>6.1</td>
<td>Sample Scenarios of Basic Relations. (a) A Sequence relation between S1 and S2 (S1 ⊳ S2). (b) A Parallelism relation between S4 and S3 ((S4 ⊕ S3) ⊳ S5). (c) A Mutual exclusion relation between S7 and S6 ((S7 ⊗ S6) ⊳ S8). (d) An Iteration relation of S9 with itself (S9*).</td>
<td>140</td>
</tr>
<tr>
<td>6.2</td>
<td>Online Ordering Scenario</td>
<td>142</td>
</tr>
<tr>
<td>6.3</td>
<td>A Sample SDG Model</td>
<td>149</td>
</tr>
<tr>
<td>6.4</td>
<td>SDG Decomposition</td>
<td>150</td>
</tr>
<tr>
<td>6.5</td>
<td>Sample Agent knowledge Base Model</td>
<td>152</td>
</tr>
<tr>
<td>6.6</td>
<td>States of Service Agent</td>
<td>153</td>
</tr>
<tr>
<td>6.7</td>
<td>A Sample Belief Knowledge Base of a Service Agent, Showing Selection Process of BestTerminalNode Function.</td>
<td>163</td>
</tr>
<tr>
<td>6.8</td>
<td>Online Ordering Scenario</td>
<td>175</td>
</tr>
<tr>
<td>6.9</td>
<td>Agent Communications of Online Shopping Scenario.</td>
<td>177</td>
</tr>
<tr>
<td>6.10</td>
<td>A Sample Belief Knowledge Base of a Service Agent</td>
<td>181</td>
</tr>
<tr>
<td>6.11</td>
<td>Number of Messages without CFPs</td>
<td>183</td>
</tr>
<tr>
<td>6.12</td>
<td>Communication Cost for Different Agent Community Sizes</td>
<td>184</td>
</tr>
</tbody>
</table>
List of Tables

1.1 Knowledge Types in Web Service Recommendation . . . . . . . . . 5
2.1 USER-ITEM RATING MATRIX . . . . . . . . . . . . . . . . . . . 12
4.1 Sample Recorded QoS Attributes . . . . . . . . . . . . . . . . . . 58
4.2 RUNNING EXAMPLE: USER-SERVICE MATRIX $W_{QoS}$ . . . . . . 62
4.3 RUNNING EXAMPLE: $W_{QoS}$ WITH BINARY RATING SCALE . . . 65
4.4 $J_{c_1}$ USER SIMILARITY MATRIX . . . . . . . . . . . . . . . . . . 67
4.5 $J_{c_2}$ USER SIMILARITY MATRIX . . . . . . . . . . . . . . . . . . 67
4.6 $J_{c_3}$ USER SIMILARITY MATRIX . . . . . . . . . . . . . . . . . . 68
4.7 $J_{c_4}$ USER SIMILARITY MATRIX . . . . . . . . . . . . . . . . . . 68
4.8 RTT MATRIX . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83
4.9 FR MATRIX . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83
4.10 Evaluation Approaches . . . . . . . . . . . . . . . . . . . . . . . . 86
4.11 Accuracy Performance Comparison (Response Time) . . . . . . . . 88
4.12 Accuracy Performance Comparison (Failure Rate) . . . . . . . . . 89
6.1 Messages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 154
6.2 Numeric Terms of Sample Services . . . . . . . . . . . . . . . . . . 163
List of Publications


Chapter 1

Introduction

1.1 Motivation

Since its beginning, the World Wide Web has become a crucial information delivery platform. Through ubiquity feature, it provides valuable benefits for business, government, and private sectors. As the Web technologies have been evolving, the Web became not only a source of information but also an infrastructure capable of providing diverse functionalities in a form of Web applications through what is called Web engineering. A powerful means that can provide such functionality is the Web service. Web services are loosely-coupled, self-contained, Web-accessible programming functions that can be published, discovered, invoked, composed, and finally executed. Web services are standardized means for diverse, distributed software applications to be published on the Web and to interoperate seamlessly. Recently, corporations are increasingly providing their services within and among organizations through intranets or the cloud. While each Web service provides its own functionality, many of them can be composed together to achieve more complex functionality, i.e., value-added service. Technically, web services
Chapter 1: Introduction

fall into two categories: information-gathering services, e.g., the weather service at www.weather.com and world-altering services, e.g., the flight-booking service at www.aircanada.com.

Web service composition (WSC) is a general task of composing component-based programs. The composition process needs the following information: computer-interpretable descriptions of the required task, the capabilities and characteristics of available Web services, and optionally certain user information or preferences. Moreover, automated composition of Web services requires that the composer system is able to automatically select, integrate, and invoke multiple Web services in order to achieve the user provided objectives. Automated WSC problem is mainly motivated by the need to improve the efficiency of composing services. Typically, available Business Process Management (BPM) systems use generic business process to represent services in order to help organizations optimize business performance. Moreover, with the advent of cloud computing, an increasing number of businesses are adopting a strategy of blending cloud services available from multiple providers.

On the other hand, E-commerce/M-commerce corporations are also adopting recommender systems that are built to respond to the ever increasing demand of nowadays users with their personalized needs. For example, the user may ask for a recommendation of an item, a service, a composite service such as a movie, car rental, or a full tourism-trip arrangement with some specific preferences. Further domain examples may include: Travel, Banking and Finance, Government, Healthcare and Life Sciences, Insurance, Retail, and Supply Chain Management. Many of these applications exploit extensive internet- or intranet-accessible data.
Chapter 1: Introduction

However, having the feedback and rating information stored solely in corporation servers, in addition to items/services profiles makes performing such recommendation, integration and interoperation manually infeasible, costly and time consuming. In addition, considering the popularity of mobile devices, their limited capabilities and connectivity issues, besides the large number of services available on the web nowadays, it is crucial to provide assistance to current users who are mainly mobile when seeking for matchable items/services according to their needs.

In this thesis, the aim is to advance the research on Web service recommendation by proposing a recommendation framework that is capable of not only recommending an individual Web service but also a composite one when no service available to fulfil the user request. The framework consists of two main parts. First, a recommendation model for individual Web service is proposed where the QoS profile is considered as an implicit rating scheme. Second, a composite service model is presented in which AI-based planning using Agent technology is adopted to dynamically and flexibly construct composite service workflow.

The rest of the chapter presents the research problem that explains the research questions, the research assumptions, the key contributions and the thesis outline.

1.2 Research Problem

The goal of this research is to design and implement a generic recommendation framework for web services. In web service recommendation, there are two consumption cases: individual and composite. A composite web service constitutes of two or more individual web services that can be invoked in a certain order.

In recommendation techniques, web services can be processed as items. However, web services have additional types of information that need to be considered
for providing valuable recommendation results. In other words, items and web services share certain kinds of information such as category, price, and ratings, while only web services have QoS properties such as response time and failure rate. The high level of similarity between items and web services makes possible to process them with the same recommendation approaches.

The consumption of web services has several problems that make it impractical for a user to find an appropriate web service in a timely manner. Therefore, to assist users in finding the best services according to their needs, using recommender systems can be an appropriate solution. To specify why it is difficult for an ordinary user to find and select an appropriate web service, a set of major problems is listed as follows:

1. Huge number of services exist to select from, over the Internet.
2. Several search engines and service crawlers can be used to search for web services.
3. Users spend much time to search and select services that may fit their needs.
4. Service functionality and quality cannot be guaranteed.
5. When no individual service is available, a composite workflow-based service needs to be generated.

Furthermore, information usually involved in recommender systems can differ significantly in terms of their obtaining methods, sources, and data types. For web service recommendation, approaches may utilize explicit or implicit knowledge. Further, the knowledge can reside on either provider side or consumer side. The type of knowledge can also be one of the following: service description profiles (e.g., WSDL), semantic description profiles (e.g., OWLS), QoS observations, user
ratings, user behaviour (usage data), user profile, etc. Table 1.1 briefly classifies the types of knowledge that can be used in web service recommendation.

In this work, the intention is to design a generic recommendation framework that can take into account a variety of the above knowledge types in the recommendation process. As a result, the framework will not be restricted to a specific type of knowledge and therefore it has higher productivity and flexibility. To address this research problem, the following main question needs to be answered: How to recommend individual/composite web services based on users’ requests? Furthermore, several secondary questions also need to be addressed in order to form a full answer to the main question, as in the following:

1. What type of knowledge is required?
2. How to represent the knowledge?
3. How to deal with incomplete recommendation data?
4. How to recommend appropriate individual services to the end user?
5. How efficient is the proposed model?

<table>
<thead>
<tr>
<th>Obtaining methods</th>
<th>Source of knowledge</th>
<th>Type of knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>explicit</td>
<td>Provider</td>
<td>service description profiles (e.g., WSDL)</td>
</tr>
<tr>
<td>explicit</td>
<td>Provider</td>
<td>semantic description profiles (e.g., OWLS)</td>
</tr>
<tr>
<td>explicit</td>
<td>Provider</td>
<td>QoS observations</td>
</tr>
<tr>
<td>explicit</td>
<td>Consumer</td>
<td>user ratings</td>
</tr>
<tr>
<td>explicit</td>
<td>Consumer</td>
<td>user profile</td>
</tr>
<tr>
<td>implicit</td>
<td>Consumer</td>
<td>user behaviour (usage data)</td>
</tr>
</tbody>
</table>

Table 1.1: Knowledge Types in Web Service Recommendation
Chapter 1: Introduction

6. Can the model be applied to other recommendation domains?

7. How to recommend composite web services in a complexity-balanced manner?

8. How effective is the proposed composition approach?

This thesis proposes a recommendation framework not only for individual Web Services but also for composite Web services, towards the establishment of a comprehensive service recommendation framework. Based on the integration of similarity-based graph with user-service recommendation model and also of a distributed AI-based planning algorithm with Multi-Agent technology, the framework is able to recommend individual and composite web services to end users, respectively.

1.3 Thesis Assumptions

In this thesis, the following general assumptions are made in order to ensure that the thesis focuses on the research aim and research questions outlined.

1. It is assumed that all available services are discovered and ready for access from a centralised repository. If no services are matched during recommendation or composition process, the required services are assumed to be unavailable.

2. It is assumed that all implicit and explicit rating information such as QoS attributes are recorded and extracted prior to performing recommendation task. Also the recommender system is responsible to continuously maintain an updated records of services’ invocations.
3. It is assumed that all services with their profiles are distributed and assigned
to service agents acting as formal delegates for these services.

4. It is assumed that the service dependency relationships are also provided
for each service agent prior to composition and that if a new service/agent
joined the community, it must inform all agents with its capabilities and
availability.

1.4 Contributions

The main contributions of this research are listed as follows. The detailed evaluation of these contributions is discussed in Chapters 4, 5, and 6.

- Development of an individual Web Service recommendation model using
Graph theory with Random Walk algorithm (Chapter 4). The model utilizes
the Jaccard coefficient in several variants to create two Unipartite similarity-
based graphs that capture similarities among Users and among Services. By
integrating these graphs with the original user-service rating graph, a richer
recommendation model is constructed. Using the Top-K Random Walk rec-
ommendation algorithm, a final set of recommendations is delivered to end
user. The model proves its well-behaviour in terms of sparsity tolerance and
recommendation accuracy.

- Reduction of individual service recommendation model in terms of time
complexity (Chapter 5). To minimize complexity of the proposed model, a
thresholding technique is proposed in which Random Walk algorithm is bet-
ter guided using a reduced subset of users based on their Jaccard similarities,
rather than the entire set. Furthermore, the applicability of the proposed
model as a generic recommendation model is also examined through the use of an ordinary rating domain.

- Presentation of a service composition model based on Agent-technology (Chapter 6). In this model, a distributed service dependency model based on AND/OR graph structure is decomposed and distributed among individual members of the Agent community. The agents are equipped with a well-defined internal reasoning mechanism based on agents’ knowledge. Through a proposed communication protocol, the agents collaborate to find a cost-effective executable workflow according to end user request.

- Feasibility and effectiveness demonstration of all components of the proposed framework, using publicly available datasets, a recommendation library, and a multi-agent platform (Chapters 4, 5, and 6).

In addition, earlier versions of several parts of this thesis have been published in international conferences and journals. These include the proposal of the Integrated-Model QoS-based Graph for Web Service Recommendation at the IEEE ICWS, 2015 [1], the reduced similarity based model for web service recommendation at the IJSC, 2016 [2], the Agent-Based Model to Web Service Composition at the IEEE SCC, 2016 [3]. Some other publications, which are closely related to this research but not directly addressed in this thesis, can also be found in the publication list.
Chapter 1: Introduction

1.5 Thesis Outline

The thesis is organised as follows:

**Chapter 2** presents the background of this research including essential recommender system models and approaches where their strengths and limitations are highlighted.

**Chapter 3** provides an overview of the literature on current web service recommendation models whether of individually- or compositely-oriented categories.

**Chapter 4** proposes a Model for individual Web Service recommendation approach using Graph theory with Random Walk algorithm. The model integrates both user-service ratings with users and service similarities using Jaccard coefficient to benefit from tighter relationships among users and services.

**Chapter 5** discusses how the proposed recommendation model is improved in order to minimize its complexity. Before applying Random Walk algorithm, the model is reduced such that only a selected subset of users is considered based on Jaccard similarities, while observing the recommendation accuracy.

**Chapter 6** presents a service composition recommendation approach in which an AI planning Agent-based service Model is proposed where Agents collaboratively cooperate through internal reasoning and external communication to attempt finding a cost effective workflow solution to current composition request.

**Chapter 7** summarises the contributions of this thesis and proposes future research works.
Chapter 2

Background

Service Oriented Architecture (SOA) is an emerging technology that enables decoupled applications running on different machines to efficiently exchange data, without the need of additional third-party software or hardware. With the immense increase of adopting this technology, the need for automation in selecting and recommending web services is evident. At this point, the role of recommender systems comes in, where they are applied to assist the user to choose one or more web services according to their preferences. Typically, the selection process utilizes functional and/or non-functional characteristics of services.

This chapter presents an overview of background information on recommendation approaches. First, it introduces the recommender systems. Then, it presents the most common recommendation techniques, highlighting their advantages and disadvantages. Finally, advanced recommendation techniques are briefed including dimensionality reduction methods and Graph based methods, on which the main part of this work is based.
2.1 Introduction to Recommender Systems

Recommender Systems (RSs) are defined as software applications and tools that provide recommendations to users for various items. Item in recommender systems is a general term which denotes what a system recommends to users [4]. Depending on the type of recommendation needed, the item may be any product or service such as books, music, movies, news, and so on. In fact, personalization plays an essential role in the recommendation process. By utilizing personalization, recommender systems suggest to user products and services that they might be more interested in with less effort. Although there is significant research done in both academia and industry in this area since the mid-1990s, there is still tremendous need and space for improvements and optimizations in recommendation techniques due to the ever-growing amount of information in the digital world [5].

2.2 Recommendation Techniques

The main goal of recommendation techniques is to estimate ratings for items that are not previously used or rated by a user. Usually, this predicting process is based on previous ratings of the user. Once the unknown ratings are estimated for a given user, the items with the highest predicted ratings can then be recommended. To perform this task, recommendation techniques need an input which is usually provided as a user-item rating matrix, capturing users’ preferences. Table 2.1 illustrates subsets of such an input to a recommendation system.

Following [5], a formulation of the recommendation problem is that: Let \( C \) be the set of all users, and \( S \) be the set of all items. Both user and item spaces can be enormous, for example, millions of users and items in recommender systems for movies, DVDs, songs or news, etc. Let \( u \) be a utility function measuring the
Chapter 2: Background

Table 2.1: USER-ITEM RATING MATRIX

<table>
<thead>
<tr>
<th></th>
<th>item1</th>
<th>item2</th>
<th>item3</th>
<th>item4</th>
</tr>
</thead>
<tbody>
<tr>
<td>user1</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>user2</td>
<td>-</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>user3</td>
<td>4</td>
<td>-</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>user4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>-</td>
</tr>
</tbody>
</table>

usefulness of item $s$ to user $c$, i.e., $u : C \times S \rightarrow R$, where $R$ is a set of real or non-negative integers within a certain range which is usually represented by ratings (e.g., 1-5 rating scale). Then, for each user $c \in C$, the goal of the recommender system is to find $s' \in S$ that maximizes the user’s utility.

Different techniques from approximation theory, various heuristics, and machine learning are used to estimate unknown ratings. However, according to rating estimation approach, recommender systems are classified into Content-based, Collaborative, and Hybrid approaches [5].

2.2.1 Content-based Methods

The idea of the content-based approach is based on the similarity of contents between two items. Therefore, to recommend item $s$ to user $c$, the similarity between the content of $s$ and other items previously rated by $c$ is measured. For example, if news items related to economics are highly rated by user $c$ in a news recommender system, then other news items in economics receive a higher similarity score. In content-based approach, each item is represented by a profile in which its properties are stored. Usually, the properties are extracted from the item’s content before starting to calculate the similarities. Common examples of this kind are text-based items such as articles, books, and news. In the following, a formulation of content-based recommendation techniques is presented [5]:

12
The utility function $u(c, s)$ is defined as $u(c, s) = \text{score}(\text{Profile}(c), \text{Content}(s))$, where $\text{Profile}(c)$ is the profile of user $c$, $\text{Content}(s)$ is the profile of item $s$. Both profiles contain keywords representing user’s preferences and item’s characteristics respectively. These profiles are usually obtained either explicitly or implicitly (i.e., by examining the content of previously rated items by $c$) using keyword analysis techniques from information retrieval.

Following the above formulation, it is necessary to measure the importance of words (i.e., to select keywords) to effectively represent both user and item profiles. Let us assume that each user or item profile can be represented by a document $d_j$. Using weighting measure $w_{i,j}$, the importance of word $k_i$ in document $d_j$ is then determined. In information retrieval, one of the most popular measures for specifying keyword weights is the term frequency/inverse document frequency (TF-IDF) measure [5][6], which is:

Assume that $N$ is the total number of documents that can be recommended to users, $n_i$ is the number of documents that keyword $k_i$ appears in and $f_i,j$ is the number of times keyword $k_i$ appears in document $d_j$.

- **Term Frequency (TF)**: The term frequency or normalized frequency of keyword $k_i$ in document $d_j$, is defined as:

$$TF_{i,j} = \frac{f_{i,j}}{\max_z f_{z,j}}$$ (2.1)

Where the maximum is obtained over the frequencies $f_{z,j}$ of all keywords $k_z$ that appear in $d_j$.

- **Inverse Document Frequency (IDF)**: Let $N$ be the total number of documents and $n_i$ be the number of documents containing keyword $k_i$. IDF of $k_i$
is defined as:

$$IDF_i = \log \frac{N}{n_i} \quad (2.2)$$

Because the keywords that appear in many documents are not as informative and useful in distinguishing between a relevant and a non-relevant document, they are penalized with IDF.

- **TF-IDF**: The TF-IDF weight or importance for keyword $k_i$ in document $d_j$ is defined as:

$$w_{i,j} = TF_{i,j} \times IDF_i \quad (2.3)$$

Now, the content of document $d_j$ using its keywords or important words is defined as:

$$Content(d_j) = (w_{1,j}, ..., w_{k,j}) \quad (2.4)$$

In content-based recommendation approach, document $d_j$ may represent a user profile ($Profile(c)$) or item content ($Content(s)$) by TF-IDF vectors $\vec{w}_c$ and $\vec{w}_s$ of keyword weights. To recommend items similar to the preferred items by the user in the past, the similarity between documents can be easily measured. By considering the user profile as a document of representative keywords, and item profile as a document of important keywords, the similarity between them is then calculated. As a result, documents with the highest similarities (i.e., most similar to user preferences) will be recommended to a given user. Using cosine similarity [7], the utility function is defined as:

$$u(c, s) = \cos(\vec{w}_c, \vec{w}_s) = \frac{\vec{w}_c \cdot \vec{w}_s}{\| \vec{w}_c \|_2 \times \| \vec{w}_s \|_2} \quad (2.5)$$
Although content-based recommendation approach is effective in some kinds of recommendation applications, it has significant limitations [5][8].

- Content-based techniques need to process items that have machine readable contents, such as text-based items, to be able to parse and extract item properties and features. It is impractical to manually assign attributes to items, for instance, a video recommendation system dealing with millions of items.

- These techniques lack the ability to evaluate the quality of items. For example, in article recommendations, if two articles belong to the same subject (i.e. have similar keywords), content-based recommendation methods cannot prioritize them based on the quality of writing, i.e., the well-written vs. the poor one.

- Over-specialization problem, in which users are only restricted to recommendations related or similar to the items they preferred in the past. That means they never get an opportunity to receive recommendations of items in other tastes which they might be interested in.

- Finally, new user problem is another constraint of content-based techniques. The problem arises when a user first joins the system and has no sufficient number of ratings yet. That is because a system is unable to understand user’s taste and preferences so that it can recommend relevant items to him.

2.2.2 Collaborative Filtering Methods

Collaborative Filtering (CF) methods recommend items liked or preferred by other users who have similar preferences to a given user. To overcome the limitations of
content-based methods, both academia and industry widely adopted CF methods. In fact, it is the most common approach of recommendation techniques. Formally, the estimation of utility $u(c, s)$ of item $s$ for user $c$ is defined based on the utilities $u(c_j, s)$ assigned to item $s$ by all similar users $c_j \in C$ to user $c$ [5].

The first works in “automation” of prediction in collaborative filtering techniques are Ringo [8], GroupLens [9], and Video Recommender [10]. In October 2006, Netflix, an online DVD rental service, launched an open competition (Netflix Prize) for the best collaborative filtering algorithm. Thousands of scientists, engineers and students have been provided with a real industrial data set of 100 million movie ratings for the first time, aiming at developing an optimized collaborative filtering algorithm. In September 2009, Koren and Bell [11] achieved the highest improvement by 10% over the original Netflix algorithm.

According to [12] collaborative filtering approaches can be classified into two general categories: memory-based (or heuristic-based) and model-based techniques. The following subsections describe each in details.

2.2.2.1 Memory-based CF Methods

Memory-based CF methods exploit the user-item rating data to measure similarity between users or items. Typical examples of this class are User-based and Item-based collaborative filtering approaches. In user-based approach, the value of an unknown rating $r_{c,s}$ by user $c$ for item $s$ is calculated as an aggregate of the ratings by other users (mostly similar users to user $c$) for item $s$ [5]:

$$r_{c,s} = \text{agg}_{\hat{c}\in \hat{C}} r_{\hat{c},s}$$ (2.6)
where $\hat{C}$ is the set of $N$ users that are most similar to user $c$ and who have rated item $s$. The simplest aggregation function that can be used is the average:

$$r_{c,s} = \frac{1}{N} \sum_{\hat{c} \in \hat{C}} r_{\hat{c},s} \quad (2.7)$$

The most common aggregation approach is to use the weighted sum:

$$r_{c,s} = k \sum_{\hat{c} \in \hat{C}} \text{sim}(c, \hat{c}) \times r_{\hat{c},s} \quad (2.8)$$

where $\text{sim}(c, \hat{c})$ is the similarity between users $c$ and $\hat{c}$. The limitation of weighted sum is that it does not take into account the fact that different users may use different rating scales. To overcome this limitation, the deviations from the average rating of users, $\overline{r}$, is considered by using adjusted weighted sum $[9][13]$:

$$r_{c,s} = \overline{r_c} + k \sum_{\hat{c} \in \hat{C}} \text{sim}(c, \hat{c}) \times (r_{\hat{c},s} - \overline{r_c}) \quad (2.9)$$

where $k$ is a normalizing factor and is usually calculated as:

$$k = \frac{1}{\sum_{\hat{c} \in \hat{C}} |\text{sim}(c, \hat{c})|} \quad (2.10)$$

There are various approaches to measuring the similarity $\text{sim}(c, \hat{c})$ between users or between items in collaborative filtering systems. The similarity between two users is based on their ratings of items, co-rated by both users; while the similarity between two items is driven by the ratings given by users to them. The two most popular approaches are correlation-based and cosine-based similarity calculation methods.
• **Correlation-based approach:** Assume $S_{cc'}$ to be the set of all items rated by both users $c$ and $c'$, then in the user-based correlation approach, the Pearson correlation coefficient \[8\][9] is used to measure the similarity between users $c$ and $c'$ as follows:

$$
sim(c, c') = \frac{\sum_{s \in S_{cc'}} (r_{cs} - \bar{r}_c)(r_{c's} - \bar{r}_{c'})}{\sqrt{\sum_{s \in S_{cc'}} (r_{cs} - \bar{r}_c)^2 \sum_{s \in S_{cc'}} (r_{c's} - \bar{r}_{c'})^2}} \tag{2.11}
$$

• **Cosine-based approach:** In the cosine-based approach \[12\][13], the two users $c$ and $c'$ are considered as two vectors in $n$-dimensional space, where $n = |S_{cc'}|$:

$$
sim(c, c') = \cos(\vec{c}, \vec{c'}) = \frac{\vec{c} \cdot \vec{c'}}{\|\vec{w}_c\|_2 \times \|\vec{w}_{c'}\|_2} \tag{2.12}
$$

Where $\vec{c} \cdot \vec{c'}$ shows the dot-product between two vectors.

Despite the popularity of user-based collaborative filtering approach, there are some significant challenges such as *scalability* and *quality* of recommendations. Calculating similarities between millions of users nowadays is impractical in real-time. On the other hand, reducing the number of candidate neighbours by some method assists in increasing the computational performance, yet diminishing the quality of recommendations. Therefore, a trade-off is still required, between reducing data to speed up these systems and the resulting quality of recommendations.

### 2.2.2.2 Model-based CF Methods

Memory-based approaches consider the entire user-item rating matrix to predict ratings of unknown items. In contrast, model-based approaches use the collection
of ratings to learn a model using statistical and machine learning techniques, which is then used to predict unknown ratings. Authors in [12][14] conduct a study to compare their respective model-based approaches with standard memory-based approaches. They concluded that model-based methods outperform memory-based approaches in terms of accuracy of recommendations in some applications. However, their comparisons are not theoretically supported.

In summary, collaborative filtering methods have substantial advantages over classical content-based methods. They can recommend items based on the items highly rated by most similar users. Besides they are not limited to text based items or a certain category of them previously liked by the user. However, collaborative filtering methods still suffer from certain problems.

New user and new item are common problems. The first problem occurs because there are no sufficient ratings by the new user to understand user preferences when looking for most similar users to him. On the other hand, the new item problem occurs when a new item is added to the system. At the time, the system lacks a sufficient number of ratings for the new item to be able to recommend it. One way to address these two problems is to apply hybrid methods which are described in the next subsection.

Sparsity is another common problem associated with collaborative filtering methods. In today’s systems, although millions of users and items can be represented with a large matrix, the number of available ratings is very small compared to the number of unknown ratings. In other words, items with only a few ratings are not considered to be recommended regardless of their rating’s value. In addition, users with unusual tastes cannot be matched with other users according to preferences [15].
Chapter 2: Background

To overcome the sparsity problem, researchers in [16] proposed an extension to traditional collaborative filtering techniques, called ‘demographic filtering’. In this method, the similarity of measurement between users is based on the users’ profiles, in addition to their common ratings. For instance, users’ location, occupation, education, gender, and age, with similar ratings for co-rated items are all considered for similarity measurements.

2.2.3 Hybrid Methods

*Hybrid methods* combine collaborative and content-based methods to overcome certain limitations of each of these methods [15][16]. There are different ways to combine collaborative and content-based approaches into a hybrid recommendation system categorized as:

1. Prediction results of different recommendation approaches carried out separately are combined.
2. Incorporating collaborative features into a content-based approach or vice versa,
3. Applying a general unified model that combines both content-based and collaborative characteristics [5].

2.2.4 Advantages and Disadvantages of CF Approaches

This section discusses the pros and cons of the CF approaches [17]. The CF approaches are neighbourhood approaches based on rating correlation. They have some advantages such as:
• **Simplicity**: Intuitively, Neighbourhood-based methods are relatively easy to implement. Only one parameter, which is the number of neighbours considered in the prediction, requires tuning.

• **Justifiability**: They provide a concise justification for the generated predictions. For instance, in item-based approach, the list of neighbor items along with their associated ratings, are usually presented to the user as a justification. The benefits are twofold: it assists the user to understand the relevance of the recommendations better, and it could also support interactivity feature of the system where users can pick the neighbors of higher importance to them to respect in the recommendation [18].

• **Efficiency**: Neighbourhood-based systems are efficient. Unlike model-based, they require no costly training phase, which needs to be frequently performed in real-world commercial applications. To reduce the cost of recommendation phase, the nearest neighbors can be pre-computed offline providing proximal instantaneous recommendations. Furthermore, the reduced amount of memory needed to store nearest neighbour lists enable these approaches to scale to large applications, with millions of users and items.

• **Stability**: Being non-significantly affected by the constant addition of users, items, and ratings, (typically accompanying large commercial applications), offers a high degree of stability to the system. For example, an item-based system needs not to re-train with in responding to new users’ requests. Moreover, only the similarities between a newly added item and the existing ones need to be computed.

Along with the above advantages, the neighbourhood approaches have two main flaws, which may limit their effectiveness:
Chapter 2: Background

- **Limited coverage**: Since the similarity between two users is based on comparing their ratings for the same items, users are neighbors only if they have rated some common items. This assumption is strongly delimiting the opportunity of users who have rated only a few or no common items. Also, this excludes the items that are unrated by neighbors from recommendation, i.e., lack of coverage.

- **Sensitivity to sparse data**: The accuracy of neighbourhood-based recommendation approaches suffers from the lack of availability of ratings. In fact, users typically rate only a small proportion of available items, causing the Sparsity problem [19]. Furthermore, this leads to the cold-start problem [20] where users or items newly added to the system may have no ratings at all. With sparsity, two users or items are unlikely to have common ratings. Consequently, in neighbourhood-based approaches, a very limited number of neighbors will be considered in the recommendation. Moreover, biased recommendations can also be experienced since similarity weights may be computed using only a small number of ratings.
2.3 Advanced Recommendation Techniques

The disadvantages discussed in the previous section have led to the emerge of another type of recommendation techniques, namely: dimensionality reduction and graph-based methods. The following subsections give a brief description of each.

2.3.1 Dimensionality reduction methods

The idea of Dimensionality reduction methods is that, in order to deal with the problems of limited coverage and sparsity, users and items are projected into a reduced latent space where the most notable features can be captured. In particular, users and items are compared in a subspace of high-level features, enabling the discovery of more meaningful relations. Accordingly, even if two users have no common rated items, this approach can discover a relation between both, which results in less sensitivity to sparse data. Essentially, within recommender systems, dimensionality reduction are achieved in two ways: 1) decomposition of a user-item rating matrix [21] [22], and 2) decomposition of a sparse similarity matrix [23].

2.3.2 Graph-based Methods

In graph-based approaches, the graph structure is used to represent the rating data where nodes can be users, items or both, while edges encode the similarities/interactions between the users and items. In its simplest model, the data is modeled as a bipartite graph where the two sets of nodes represent users and items, and an edge connects user $u$ to item $i$ if $u$ has a rating for $i$ in the system. The edge can also be weighted by a value of its corresponding rating. Also, the
nodes can represent either users or items while an edge connects two if the ratings corresponding to these nodes are sufficiently correlated.

In such models, correlation prediction can be used to estimate the rating of a user $u$ for an item $i$ based only on the nodes directly connected to $u$ or $i$, as in standard approaches. However, Graph-based approaches can include nodes that are not directly connected through allowing the propagation of information along the edges of the graph. Greater weights allow more information to pass through (property of propagation). Besides, the influence of a node on another is negatively correlated with the length of the path between them in the graph (property of attenuation) [24] [17]. In the following section, two of widely used graph based recommendation approaches on graphs, namely: Random Walk and ItemRank approaches are presented.

2.3.2.1 Random walk similarity

Random Walk [25] is defined as a Markov chain that describes a sequence of nodes visited by a random walker. If a random walker in state $i$ at time $t$, then a random variable $s(t)$ contains the current state of the Markov chain at time step $t$, such that $s(t) = i$.

In graphs, transitive associations in graph-based methods can also be defined within a probabilistic framework, in which the similarity between users or items is evaluated as a probability of reaching these nodes in a random walk. Since the transition probabilities are based only on the current state (i.e., not on the past ones), it is considered a first-order Markov process. Formally, this can be defined by a set of $n$ states and an $n \times n$ transition probability matrix $P$ such that the probability of jumping from state $i = s(t)$ to $j = s(t+1)$ at any time-step $t$ is:
Denote $\pi(t)$ the vector containing the state probability distribution of step $t$, such that $\pi_i(t) = Pr(s(t) = i)$, the evolution of the Markov chain is characterized by $\pi(t + 1) = P^T \pi(t)$ with $\pi(0) = \pi^0$ and where $T$ is the matrix transpose. Moreover, under the condition that $P$ is row-stochastic, i.e. $\sum_j p_{i,j} = 1$ for all $i$, the process converges to a stable distribution vector $\pi(\infty)$ corresponding to the positive eigenvector of $P^T$ with an eigenvalue of 1. This process is often described in the form of a weighted graph having a node for each state, and where the probability of jumping from a node to an adjacent node is given by the weight of the edge connecting these nodes.

Due to the connectivity of the graph, the Markov chain is irreducible, meaning every state can be reached from any other state. Otherwise, the Markov chain can be decomposed into independent closed subsets of states (i.e., no relations between them), where each closed subset are irreducible, on which the random walk can be independently applied. For more details on Markov chains, the reader is invited to consult standard textbooks on the subject (e.g., \cite{26}, \cite{27}) \cite{17} \cite{28}.

### 2.3.2.2 Itemrank

Itemrank \cite{24} is based on the PageRank algorithm for ranking Web pages \cite{29}. In this approach, preferences for new items of user $u$ are ranked as a probability of $u$ to visit $i$ in a random walk on a graph $g$. In $g$, nodes correspond to the items of the system, while edges connect those items that have been rated by common users. A transition matrix $P$ holds the edge weights, such that $P = |I| \times |I|$, for which $p_{i,j} = |U_{i,j}|/|U_i|$ is the estimated conditional probability of a user to rate
an item $j$ if it has rated an item $i$. Following PageRank, at any step, the random walk can either jump using $P$ to an adjacent node with fixed probability $\alpha$, or "teleport" to any node with probability $(1 - \alpha)$. Let $r_u$ be the $u_{th}$ row of rating matrix $R$, the vector $du = ru/||ru||$ denotes the probability distribution of user $u$ to teleport to other nodes.

Based on these definitions, the state probability distribution vector of user $u$ at step $t + 1$ is recursively defined as

$$
\pi_u(t + 1) = \alpha P^T \pi_u(t) + (1 - \alpha)d_u
$$

(2.13)

Usually, $\pi_u(\infty)$ is computed by first initializing the distribution as uniform, i.e. $\pi_u(0) = \frac{1}{n}1_n$, and then iteratively updates $\pi_u$ until convergence. Once $\pi_u(\infty)$ is calculated, the system recommends to user $u$ the item $i$ with the highest $\pi_{ui}$.

### 2.4 Summary

This chapter presented the background information on recommendation approaches. It first overviewed some of the most popular approaches used in recommender systems, including content-based, collaborative filtering and hybrid methods. As collaborative filtering methods are widely used, their strengths and limitations are also presented with an insight to address some of their limitations in this thesis. The problem addressed in the adopted approach is one of a larger set of problems that make up the Web service recommendation problem.

Then, the chapter also presented an overview of more advanced recommendation techniques including dimensionality reduction methods and Graph based methods, on which the main part of this work is based.
Chapter 3

Literature Review

Recently, many research works have been conducted on effective utilization of web services, as a powerful means of communication among heterogeneous systems over the ever-expanding internet network. This chapter explores previous works in web service recommendation and web service composition techniques. In the first, the chapter focuses on categorizing those techniques in terms of type of data they utilize along with the underlying recommendation approach. While in web service composition, it concentrates on the AI Planning Composition approach, presenting previous related works. Besides, this chapter presents a group of recent sample research projects, which integrate AI Planning service composition with Agent technology in an attempt to meet nowadays computing trends, such as smart cities, human-computer-interaction, and Internet of Things. Finally, a summary of those works is presented, demonstrating the need of continuing work to push forward the advancements in this area.
Chapter 3: Literature Review

3.1 Web Service Recommendation

Although a web service can be considered as an item in recommendation approaches, they still have significant differences that should be taken into account when performing certain recommendation tasks. One difference is that Web services have more complicated representational profile than simple products. For example, a movie profile may contain simple textual information such as title, genre, a list of participating actors, production year. In contrast, a Web service profile contains functional properties (Web service operations, inputs, outputs). Further, since the Web service is a piece of code that is usually hosted in a remote server, availability and performance features, remain under no control to users. Therefore, to provide reliable and satisfying recommendations, a recommendation system needs to address a set of non-functional properties (such as quality of services, price, and execution order). It is noticeable that the term Web Service discovery is used interchangeably with WS recommendation especially when taking into account only functional properties during the search process.

In Web service recommendation, many solutions have been proposed to assist users in retrieving their interested services. However, several problems have been identified and tackled such as service selection, service execution, service recommendation, and web service composition. Existing approaches analyze different aspects of Web services, such as service descriptions, execution conditions, and service behaviours. Both functionality properties and non-functionality properties have also been examined. Several Web service presentations such as XML-based, semantic web, graph-based, have also been explored.
3.2 Prior Work on Web Service Recommendation

In order to recommend the most relevant web services matching certain user’s needs, different approaches have been proposed. They exploit different types of information, i.e., web service descriptions, QoS of services, semantic concepts, and usage patterns. In this section, brief descriptions of some of the existing works under each group are given.

3.2.1 Text-based Approaches

In one of the earliest works, Dong et al. [30] proposed a similarity search engine for Web services. The engine analyzes Web service descriptions to extract similarities between inputs, outputs, and operations of a given set of services. It first splits the names of inputs, outputs and operations then cluster them in different concepts. Each input/output is represented by a vector of three elements $i = (p_i; c_i; op)$, where $i$ is the input, $op$ is a Web service operation, $p_i$ is a set of input parameter names, and $c_i$ is a set of concepts associated with the parameter names. Whilst, each operation $op$ is denoted by a vector $op = (w; f; i; o)$, where $w$ is the textual description of the Web service, $f$ is the textual description of $op$, $i$ and $o$ identify input and output parameters respectively. To estimate the similarity between inputs, outputs or Web service descriptions, TF-IDF measure is computed between their corresponding concepts. Finally, the similarity of Web service operations is computed based on the similarities of vector elements.

Similarly, Platzer et al. [31] proposed an approach to match user’s query string with Web service descriptions. First, they collect Web service descriptions in WSDL format from different resources, such as user’s uploading, links from
websites, or references from UDDI (Universal Description, Discovery, and Integration) repositories. Next, each Web service description file is parsed to generate a description vector whereby each value indicates the number of times that the word appears in the description. Likewise, user’s query string is represented as a vector. After weighting each term by TF-IDF, the similarity between a query-vector and a document-vector is then computed by Vector Space Model (VSM). Finally, the system recommends the Web services whose descriptions have the highest similarity values with a given query string. Figure 3.1 shows the system architecture.

Although the proposed approach is based on the similarity between user’s query string and Web service description, it directly matches the query vector with document vectors without clustering input, output and operation vectors under their concepts like Dong et al. [30].

Different from [30] and [31], Blake et al. [32] attempted to utilize user’s daily routine when recommending Web services. First they capture the text data that
Chapter 3: Literature Review

a user is working on such as HTML files, Word documents, File systems, messages, such as SOAP (Simple Object Access Protocol) messages. Second, from the captured data, they extract text strings, which are compared to the operations of available Web services in the third step. They captured four naming tendencies that software designers/developers used to name a Web service. To compute similarities, they applied *Levenshtein Distance* (LD) and *Letter Pairing* (LP). In the last step, Web services with highest similarity values are recommended to a user.

Although text-based approaches show an elegant and practical approach to Web service selection task, they still suffer from several significant shortcomings as follows:

1. Polysemy problem in which one word can have many meanings
2. Synonym problem which means one meaning can be expressed by different words.
3. Web service description accuracy. It is not guaranteed that service description text is always correct and meaningful.
4. User input validity. This approach cannot flexibly address cases when a user either types incorrect words or abbreviated words in the query.
5. Limited supported languages in Web service documents. As most Web service documents are currently described in English, users are also restricted to write their queries in English. Using translation tools to switch to English may add another layer of inaccuracy to user’s query and therefore it is not recommended in Web service recommendation.
3.2.2 QoS-based Approaches

QoS features in Web service recommendation are widely addressed in the literature, as well as Web service search engines, Web service recommender system, Web service reliability prediction, Web service selection, optimal service composition and so on.

In [33], Zhang et al. proposed a search engine (WSExpress) that considers both functional and non-functional features of Web services. In functional features, they represent each Web service description by an operation, input, and output. While non-functional features are identified by QoS attributes, such as price and response time, consequently, a user’s query contains both functionality and QoS requests. The proposal defined QoS query as $QoS = (Constraint, Weight)$, in which ‘Constraint’ is a vector of queried QoS attributes. In contrast, the ‘Weight’ vector contains the weight of each QoS attribute in a user’s query. Once a user’s query is submitted, both the functionality matching and the QoS matching are then computed. Finally, to balance both sides, $\lambda \in [0, 1]$ parameter is used to generate the final search results. Figure 3.2 shows the system architecture.

While Zhang et al. [33] took into account both functionality and non-functionality
concepts in their approach, the authors in [34] considered only QoS values of Web services to build a Web service Recommender System. Initially, they built a user-item matrix, where rows represent user IDs and columns represent Web service IDs. Each entry is a vector of QoS values that are observed by the corresponding user for the corresponding Web service. The QoS metrics are round-trip-time and failure-rate of each Web service. Once the matrix is created, the similarity between Web services and users using Pearson Correlation Coefficient is then computed. The probability that user $a$ will probably use a Web service $i$ is finally formulated based on the computed similarities. To enrich the recommendation process, the authors in [35] extended the prediction function used in [34] to a hybrid one that consists of item-based and user-based prediction sub-functions. To control both sub-functions, a parameter $\mu$ is added to balance the weight of item-based and user-based prediction functions.

Similarly, authors in [36] base their proposed approach on a User-Web service QoS matrix with a prediction function. Instead of predicting user usage probability, they attempted to estimate the failure of composite Web services. The failure values are then aggregated from the failure probability of Web service components to produce a final score for a composite Web service.

Other approaches also handled QoS of composite Web services. However, their goal is to optimize Web service compositions by selecting the best Web services that satisfy given QoS constraints. The authors in [37] proposed an approach consisting of two models: a combinatorial model and a graph model. The first defines the problem as a multi-dimension multi-choice knapsack problem (MMKP), while the second model defines the problem as a Multi-Constrained Optimal Path (MCOP) problem. In both models, they use a user-defined utility function of some system parameters in order to optimize application-specific objectives.
By utilizing a Directed Acyclic Graph (DAG), the authors in [38] presented an approach that follows service execution paths in the graph. They addressed the service selection process as an optimization problem which can be solved by linear programming methods.

The authors in [39] proposed an extensible, preference-oriented, open and fair QoS model for Web service selection. Their goal is to allow providers to flexibly add new specific domain criteria for evaluating the QoS of Web service without changing the underlying computation model. In addition, it can provide means for users to accurately express their preferences without resorting to the complex coding of user profiles. Furthermore, the approach manipulates the QoS information collected from both parties, i.e., provider’s site such as service privacy and user’s site such as feedbacks.

Other approaches focus on transactional QoS criteria, such as Haddad et al. [40]. To create a composite service, a set of Web services are selected based on some transactional properties. For instance, a composite service must support atomic transactions, compensable transactions or it has to terminate after a finite number of execution steps.

In general, QoS-based approaches to Web service recommendation consume explicit knowledge, i.e., throughput, bandwidth, execution time, failure rate, privacy or feedback, etc. They can be classified into two main categories based on service’s consumption purposes: individual service demands such as [69-72], and composite service demands, such as [73-76].
3.2.3 Semantic-based Approaches

Manikrao et. al. [41] proposed a Web service recommender system based on semantic matching and rating prediction. To describe Web services, they used ontologies written in DAML (DARPA\(^1\) Agent Markup Language, recently Web Ontology Language, OWL-S). By matching all semantic attributes of Web services, they determine the similarity degree between two services while considering a given threshold. Given a user, the system can predict the rating of a Web service based on user’s ratings on similar Web services in the past. They decide that two services are more similar if their average ratings are less different.

Another work that uses DARPA Agent Markup Language to semantically describe Web services is presented in [42]. The authors attempted to address the weakness of WSDL documents in representing some semantic features. To do so, they proposed to match input/output concepts of a service request to the corresponding concepts of service advertisements. They also set specific matching degrees, namely exact, plug in, subsume and fail. Moreover, the output concepts match is given a higher priority than the input concepts match. That means, services whose advertisements are better matched on output concepts are preferred for the recommendation rather than on input concepts. Figure 3.3 shows the system architecture.

The authors in [43] applied SVD (singular value decomposition) and LSI (Latent semantic indexing) and conducted some experiments based on precision and recall metrics. Particularly, they represented Web service documents in a matrix of documents and terms. Then, they decomposed the document-term matrix to an approximate matrix using SVD. Finally, to find the closest documents to a

\(^1\)Defense Advanced Research Projects Agency
given query, the query terms and the document rows in the decomposed matrix are matched.

In their proposed approach, Paliwal et al. [44] start by linking a Web service request and Web services descriptions to high-level ontology concepts, before decomposing and matching them. The steps are:

1. Pre-processing service requests and determine the overall search category of Web services.

2. Based on high-level ontology concept of the Web service request, they retrieve relevant indexed service descriptions from UDDI.

3. Based on the terms of a given request, associated low-level ontology concepts are retrieved from an ontology framework.

4. With the retrieved concepts, they expand the request and transform the Web service descriptions into term-document matrices.

5. The resulted matrices are decomposed using SVD.

Figure 3.3: The Architecture of the DAML-S Matching Engine [42]
6. They match the matrices with the request vector in order to infer a final similarity score.

The authors in [45] applied AI planning techniques to help generate semantically-based Web service compositions. They use a collection of Web services and composition templates that are both described in OWL-S. First, they translate the OWL-S process descriptions and composition template to Hierarchical Task Network (HTN) domains and initial task networks, respectively. In order to compute Preference-Based Web service compositions, they use the best-first algorithm with two heuristics namely, Optimistic Metric Function (OM) and the Look-ahead Metric Function (LA). The functions are used to perform branch-and-bound pruning in order to effectively discard all unpromising nodes from the search space. Their work is aimed at enhancing the quality of compositions and the speed of generation by applying optimization techniques within the composition process.

Although using ontologies in representing Web services [41][42] provides better utilization of functional and non-functional properties, semantic-based approaches still suffer from significant limitations in reality. Specifically, due to the need of domain experts, the process of creating and publishing ontology annotated content remains time-consuming and error-prone.

Another limitation that might face rating-based solutions such as [41] is the new item problem. In which new items that have not yet rated are not recommended; in contrast, items with higher ratings are always recommended. By exploiting latent semantics hidden in Web service descriptions, other semantic-based approaches [43][44] can generate recommendations without creating ontologies or asking any effort from users.
3.2.4 Usage-based Approaches

In this scheme, the past usage data is used as a rich resource exposing users’ interests. However, there are still few attempts that take these data into account for Web service recommendation.

Birukou et al. [46–48] proposed a Web service recommender system that utilizes the past usage data. They introduced a new concept called 'Implicit Culture', in which if a user has a similar request with other members of the community, he will be suggested operations that were used by those members. Therefore, they record the usage of users together with their requests. When a new request is received, they will recommend the requester Web service operations that are used by other users with similar requests. VSM (Vector space model), TF-IDF, and WordNet-based semantic similarity are used to compute the similarities. In fact, the authors only used past usage data in their proposed rule-based theory. The similarity between requests is computed using text-based approaches. Hence, they miss the correlation between users and Web services. Also, the shortcomings of text-based approaches are met while matching users' requests.

3.3 Graph-based Recommendation

In previous sections, WS recommendation techniques are presented in terms of the type of information they handle. In this section, the focus is on the design aspect of the recommendation approach. Therefore, this section mainly presents some works that follow graph data model which our work is based on as an underline recommendation model. Usually, a graph-based recommendation approach includes two steps: constructing a graph from available data and ranking item nodes for a given user.
Chapter 3: Literature Review

Gori et al. [24] constructed a recommendation graph by connecting every two item nodes rated by at least one user. Then they used a random-walk based algorithm for scoring those nodes. The algorithm is called ItemRank, which utilizes active user’s preferences to rank item nodes. A similar work is described in [28], where several Markov-chain model based quantities are considered. The authors constructed a bipartite graph, in which similarities between any pair of nodes are computed.

Bogers et al. [49] proposed a Movie Recommendation approach using Random Walks over the Contextual Graph. Their approach uses original random walk on a graph by considering the user and item context (genre, director, actor, etc.). They presented the recommendation process as a browsing process of a user on a movie database website. In fact, their approach ignores an important factor of recommending movies which is the selection context, where a user chooses to watch a movie. In other words, different user preferences can be realized depending on where and with whom the user selects a movie. However, the author does not show any validation of the proposed approach.

Lee et al. [50] propose a bipartite graph to model the interactions between users and items. They define a recommendation factor set, $F$, in which each factor $f \in F$ is a combination of multidimensional data. Then they connect item nodes with each node that corresponds to a certain recommendation factor. Although their work utilizes contextual information, there are no clear principles of how to define such a crucial recommendation factor set $F$.

Lee et al. [51] adopted a graph-based approach to building a hybrid movie recommender system. In order to capture rich semantics, they presented a heterogeneous graph model that can store several types of edges. By extending the Personalized PageRank, they developed a new random walk based measure for node
ranking. They also introduced a new concept called path-guide, which is then used to guide the recommendation process. Lee et al. [52] presented a graph-based approach to building a generic graph-based multidimensional recommendation framework. Based on the framework, they proposed three recommendation methods: Co-occurrence Graph-based Method, Contextual Bipartite Graph-based Method, and Heterogeneous Graph-based Method. To validate their methods, they compared them with other existing methods in terms of recommendation accuracy. However, they only assessed their work regarding recommendation performance (i.e. accuracy) and ignore other criteria such as recommendation effectiveness, diversity, novelty, etc.

Finally, although using graph based approach is considered an effective technique to address the recommendation process, the construction of the graph, its nodes and edges, in addition to the ranking method of nodes, considering their direct and indirect relations, needs to be carefully dealt with. That is to ensure obtaining high recommendation accuracy of the final results along with overcoming essential recommendation problems such as data sparsity.
3.4 Web Service Composition

This section presents a review of works in service composition. Particularly, it concentrates on AI Planning based composition approach due to its wide adoption in current composition systems. The section starts with an overview of AI planning approach and then present some related previous works. In addition, to demonstrate the utilization of this approach in current computation trends, a set of current research projects is presented. Those projects effectively integrate the paradigm of AI Planning web service composition with multi-agent technology for implementing smart flexible systems capable of addressing current user needs.

3.4.1 AI Planning Composition

In AI, planning can be viewed as a search approach applied on a solution space. However, available solutions that are based on service composition and the execution of composed services are still limited \[53\]. Within Service-Oriented Architecture domain, a service is a semantic action (referred to as a plan) of an agent which can also serve as a basic building block of a construct. Executable plans are developed by starting with the abstract creation of a plan and ending with its execution which is usually a challenging process \[54\]. Moreover, additional practical challenges can also arise when creating plans at runtime \[55\]. As AI planning is a well-explored field, current automated WSC approaches often transform their services into logical representations such as PDDL (Planning Domain Definition Language) descriptions and then utilize standard AI planning algorithms in order to solve planning problems. For instance, WSPlan \[56\], OWLS-XPlan \[57\], or were presented by Sirin et al. \[58\], Akkiraju et al. \[59\], or Hatzi et al. \[60\], cf. Sabatucci and Cossentino \[61\]. These approaches follow an analogous mechanism:
• transforming service descriptions to PDDL or another logic language.

• using a planning algorithm to solve the search problem

• translating the results into a sequence of service calls.

Intuitively, following this mechanism delimits the capability of such composition approaches by the capability of the underlying logical language. Furthermore, some requirements of the modeling space of the applied language enforce explicit interpretation of different theoretic properties such as the Frame Problem or the Open World Assumption, leading to neglecting these design decisions in most works [62].
3.4.2 Prior Works on AI Planning Composition

The authors in [56] argue that the feasibility of developing one monolithic (AI-) planning solution that suits all the possible requirements of web service composition is less applicable than creating a flexible non-monolithic framework. Such a framework allows to plug in those planners that are best suited for certain planning domains and tasks; this is essential for addressing changing or new requirements of the users. Figure 3.4 demonstrates their idea of breaking down WSC into several components manageable by pluggable AI planning tools. It is the role of PlanningManager to select one of the available AI planners after receiving the WSDL files converted to PDDL by PDDLGenerator. Their approach is highly flexible and upgradable which facilitates dealing with evolving changes in user needs and systems platforms.

The authors in [57] developed an OWL-S service composition planner, called
Figure 3.6: SHOP2 based system architecture [58]

OWLS-Xplan. The planner allows for fast and flexible off-line composition of OWL-S services. The created an OWLS2PDDL converter which will generate and pass WS profiles in PDDL to a hybrid AI planner. The planner combines relaxed Graphplan FF-planner with local search and HTN based planning, in addition to a re-planning component. Figure 3.5 depicts the architecture of this planner.

The authors in [58] describe how HTN planning system SHOP2 is used with OWL-S Web Service descriptions. They developed a translation algorithm from OWL-S service descriptions to a SHOP2 domain. Afterwards, they implement a system that plans over sets of OWL-S descriptions using SHOP2 and then executes the resulting plans over the Web. Additionally, the system is capable of executing information-providing Web Services during the planning process. Finally, a second translator algorithm from SHOP2 back to OWL for final plan generated is also presented. Figure 3.6 illustrates the architecture of the SHOP2 system.
Chapter 3: Literature Review

The authors in [60] present an integrated approach for automated semantic WSC utilizing AI planning techniques. Their goal was to significantly facilitate both the WS discovery and the composition process by the incorporating semantic information. They first map a WSC problem into a standardized PDDL planning problem. For enhanced compositions, semantic information is used which is also facilitates approximating the optimal composite service when exact solutions are not approachable. Similar to previous works, external planning systems are integrated. The generated plans are assessed in terms of accuracy. A replacement mechanism is also applicable in a case of service failures. Finally, the produced composite service is encoded in OWL-S. Implementation of approach was accommodated by the development of the PORSCE II and VLEPPO systems. Two software components are developed, namely PORSCE II and VLEPPO. PORSCE II is designed for automatic transformations, results management, and semantic enhancement; while VLEPPO is a general-purpose planning system used to automatically invoke external planners when solutions for the problem is required. Figure 3.7 shows the architecture of the PORSCE II and VLEPPO systems.

Figure 3.7: Architectures of PORSCE II and VLEPPO [60]
However, a few other works exist where AI planning is directly used to tackle the problem of service composition. Fahndrich et al. [63] argue that state definition planning does not necessarily need to be converted to some logical representation, rather it can be executed directly in OWL. The authors suggest to describe either the start or the goal state and to use SWRL (Semantic Web Rule Language) to describe the effects of actions. They first propose to extend SeMa₂ service matcher by an automated service composition component using HTN planning on SWRL. Further, a heuristic component is used to evaluate the matching results and assist the search component (following best-first search approach) in guidance. The components then plugged in within a comprehensive multi-agent framework. Figure 3.8 demonstrates the architecture of the extended AI Planner.

A similar approach to avoid transformations to languages other than OWL-S, was presented by Redavid et al. [64]. In order to avoid the dependency to an underlying language, the authors propose to create SWRLDL (Semantic Web Rule Language Description Logic) rules from the input and output parameters.
of the service if the OWL-S service description provides no information about preconditions or effects. These rules can be stored in a domain ontology where initial knowledge and goals are constantly updated. After creating and storing SWRL rules in the ontology, a backward search is used in order to find a path that is able to achieve the pursued goal. This path can be considered as a composition of service calls.

A similar service-based approach was presented by Cruz et al. [65]. The approach is based on services, which are described in OWLS and which support SWRL rules. The output parameters of the services can be used for analysis or for input parameters of other services. To this end, the approach focuses on output parameters of services and not on ‘world-altering’ effects like in other solutions. They focused on the reliability of the composed service, such that the approach can suitably handle unexpected run-time events, to be in compliance with the geodata quality requirements in the messages exchanged among the composition services. However, they acknowledge that the implementation of geographic Web Services must follow service-oriented development principles, e.g. reusability, the separation of concerns, abstraction, and loose coupling. Figure 3.9 illustrates the architecture of the Geospatial Web Services composition system. Tong et al. [66] integrate agents with service composition techniques. The authors first define a formal agent model which integrates the web service and software agent technologies into a cohesive entity. Second, they present a distributed planning algorithm, namely Distributed Planning Algorithm for Web Service Composition (DPAWSC). They formalize web service composition into a graph search problem according to the dependence relations among service agents. The algorithm offers higher priority to the alternative solution with smaller length to be searched than one with larger length. To address the distributed nature of WSC, DPAWSC follows the concept of distributed decision making of the autonomous service agents.
Figure 3.9: Geospatial Web Services composition system [65]

The algorithm is then evaluated by simulation experiments where they study the results in terms of the size of final compositions along with the associated communication cost for generating them. However, the main goal of this work to minimize the cost based on local decisions within individual agents and does not consider other global factors such as the relations of the agent with its neighbors, which might have a significant potential on the final compositions. Figure 3.10 presents the Life cycle of agent-based web service composition.
3.4.3 Sample Research Projects

Current trends in computing such as smart cities, human-computer-interaction, and Internet of Things can employ the advances in service technology within different scenarios. In such scenarios, multiple techniques can be blended together in order to facilitate reaching their targeted ends. In particular, SOA is equipped with agent technology where some AI planning is applied. This section presents an overview of four current research projects where these scenarios are applied [62].

3.4.3.1 Connected Living

The goal of the Connected Living (CL) project is to create smart home. Each device is equipped with different sensors, where a software agent, representing the device, is deployed to perform different services, e.g. for turning on the heater, reading a humidity sensor or switching on a lamp. Another type of agents (called
assistants) is also employed to control such services, providing various web UIs and rules for controlling those device agents. In addition, agents can dynamically join and leave the platform at runtime. Recently, parts of the CL system passed into the IO-LITE platform\(^2\), figure ??\(^2\). Given that, the services are semantically annotated with input and output, precondition and effect, besides some QoS metrics, the agents’ services can be planned and orchestrated to more complex services according to the user’s goals.

### 3.4.3.2 MSG EUREF

Micro Smart Grid (MSG) at EUREF campus in Berlin is a project that is part of a larger group of projects in the Schaufenster Elektromobilität with the goal of finding optimal charging schedules for electric vehicles in a micro-smart grid (MSG) [68]. Currently, the MSG EUREF project is under field test at the EUREF campus where it creates charging schedules for six charging stations for electric vehicles, two local buffer storages, and a combined heat and power plant.

To describe the components of the micro smart grids, a domain model is designed. The set of components includes charging stations, storages, and vehicles, as well as abstract concepts such as prognoses, bookings, and charging schedules. A multi-stage optimization process is performed on those models in order to find an optimal schedule for charging vehicles and local storages for meeting a certain goal (e.g., a set of bookings), considering the best use of locally produced energy and low energy prices [69]. This optimization process includes four stages:

- Predicting available energy from local production, using machine learning techniques.

\(^2\)IO-LITE website: iolite.de/.
Chapter 3: Literature Review

- Assigning vehicles to bookings, vehicles to charging stations and swappable storages to vehicles.

- Optimizing the charging schedules stochastically.

- Load-smoothening using combined heat and power plants and/or local storages.

Software agents are used to representing each stage as a service. Some stages have different implementations to reflect requirements of the various sub-projects. Those services are finally orchestrated to the overall optimization process [70].

The schedule is created using an evolution strategy, a form of genetic algorithm [71]. Given an empty schedule, the algorithm repeatedly recombines ‘parent’ schedules, mutating ‘offspring’ schedules and then assessing them through simulating their charging processes. To assess quality, a weighted sum of key metrics is computed, such as total energy cost and ratio of bookings fulfillment. Schedules with the highest quality are carried over to the next generation, repeating the process until the quality converges.

3.4.3.3 IMA

A more complex project is the Intermodal Assistance for Megacities or IMA project, figure 3.11, aims to cope with the problem of integrating ever diversifying mobility services. Besides public transportation and private cars, car and ride sharing are becoming more common in large cities, and often the best option is a combination of different means of transportation.

In particular, first mobility providers register and offer their services through interfaces in response to corresponding routing requests. The system provides a

\[\text{IMA website: ima.dai-labor.de.}\]
high-level functionality, that can autonomously access, assess and utilize available services. Subsequently, intermodal routes are calculated and then proposed, taking into consideration the individual requirements and preferences of users.

As services are semantically enriched using OWLS, intermodal route calculation then becomes a typical planning problem. By accessing the distributed platform, the planner integrated a semantic service matchmaking component in order to find services that fit user’s preferences and properties, for instance, car-sharing service is excluded when the user has no driver’s license. After determining locations/stations of available mobility services, they are integrated as nodes into a graph featuring clusters which indicate potential changing locations between modes of transportation. Afterward, the system estimates the costs using an objective function considering user’s preferences, such as time, monetary costs, ecological footprint, etc. To relate preferences, they are normalized in respect to the
worst estimation for the corresponding route. Finally, using the A-star algorithm, an optimal intermodal routing solution is searched for utilizing this heuristic.

To facilitate the functionality, a distributed multi-agent approach is adopted to implemented the system, where software agents are equipped with bits of functionality. For dynamicity, new agents can be deployed to the system with no extra configuration where the inter-agent communication follow an \textit{out-of-the-box} model. As a result, providers of mobility service basically implement an agent offering the respective service, interfacing with the backend service (i.e., on their own hardware), and finally deploy such agent to IMA. Key advantages of such approach include increasing the trust and sovereignty of providers’ data while easing access to the system. Moreover, by adopting multi-agent approach, more sophisticated route-finding scenarios are also possible, for instance, software agents can act as delegated auction bidders for mobility providers and customer’s requests are negotiable.

\subsection*{3.4.3.4 EMD}

The EMD (Extendable and Adaptive E-Mobility Services) is a project that integrates Web services and Agent technologies. The goal is to develop semantic service descriptions for real-world services which can then be utilized in searching, matching, and planning stages, to facilitate reusability and orchestration to more complex processes.

In particular, both web services and agent actions are semantically annotated using OWL-S. SWRL is also used to describe preconditions and effects. OWL ontologies are derived from existing Java or EMF models. Finally, using a service template, services can then be searched for, both, at design time and at run-time by using \textit{SeMa}^2 \cite{72}. to enable orchestration to more complex services, the
matched services (or the runtime service templates) are then encoded into Business Process Model and Notation (BPMN) processes, with the ability to semantically annotating and adding to the list of available services. Once these processes are finalized, they are either realized as executable agent or directly invoked using a process interpreter agent [73]. The project support two capabilities, the matching of individual services against service template which are then manually assembled to larger processes, and the automated planning of complex service orchestrations [63], both at design and runtime. Unlike most previous works, the goal of its developers is to adopt planning directly at the OWL-S level instead of using PDDL as an abstraction which makes it more complex but more practical. Intuitively, services are constantly updated/removed/added and need not be known a priori, also their parameters can belong to any semantically described data type. This promotes higher interoperability on a semantic layer while enabling fuzzier service discovery. In addition to utilizing the EMD defined services, the developers aim at integrating it with the other projects (particularly with MSG EUREF and IMA.

3.5 Summary and discussion

This chapter presented previous works in both web service recommendation and Web service composition.

In Web service recommendation, existing approaches examine different properties of Web services, i.e., descriptions and requests, QoS, semantic descriptions and historical usage data. In general, text-based approaches suffer from synonymy and polysemy problems; QoS-based approaches focus only on realization properties, semantic-based approaches are error-prone and time-consuming and existing usage-based approaches ignore the correlation between users and web services.
Chapter 4: RW-Based Recommendation Model

In graph based recommendation approach, the construction of the graph, its nodes and edges, besides the ranking method of nodes, featuring direct and indirect relations, need to be carefully dealt with. That is to obtain high recommendation accuracy for the final results and also to overcome essential recommendation problems such as data sparsity.

In service composition, although AI planning is a well-explored area, current automated WSC approaches often follow a three step approach: (1) transform their services into logical representations such as PDDL descriptions, (2) utilize standard AI planning algorithms in order to solve planning problems and finally, (3) transform back the results into a sequence of service calls. This mechanism delimits the capability of the composition by the capability of the underlying logical language. Additionally, certain requirements of the modeling space of the applied language dictate explicit interpretation of different theoretic properties, leading to neglecting these design decisions. As a result, an AI Planning based composition approach, that is independent of generic Planning language, is greatly favourable, especially with service composition where the uncertainty of service functionality is considerably high.
Chapter 4

Random Walk based Recommendation Model for Web Services

As our goal is to design a recommendation model for web services that is effective in terms of recommendation accuracy and tolerance to a common recommendation issue (i.e., data sparsity), this chapter starts with an introduction, followed by a description of the recommendation model and finally the evaluation methodology. This chapter answers the following research questions:

1. What type of knowledge is required?

2. How to represent the knowledge?

3. How to deal with the lack of recommendation data?

4. How to recommend appropriate individual services to the end user?
4.1 Introduction

Service-Oriented Computation (SOC) has recently emerged as a promising Web paradigm in different application fields such as e-commerce, enterprise application integration, etc. About 30000 Web services are available on the Internet from around 8000 providers, as of June 2013 [74]. With these growing numbers, assisting service users to select appropriate Web services, that fulfill their needs, becomes difficult and time-consuming. Hence, the need for reliable Web service recommender systems is attracting more researchers, aiming at designing effective and applicable recommendation models.

Web service RSs are primarily divided into two types, functional (e.g., name, input, and output) and non-functional (i.e., Quality of Service) based systems. The quality of Service attributes includes <Response Time (RT), Failure Rate (FR), Throughput (TH), Availability (AV), Price (PR), etc.>. Most prior works in QoS-based WS recommendation utilize Neighbourhood-based Collaborative Filtering [34, 35, 75]. In which similarities among users/services are computed to predict missing QoS values to an active user by considering a set of nearest similar users/services.

They are divided into three types, user-based [75], item-based [76] and hybrid [34, 35, 77, 78] approaches. In User-based approaches, ratings for active users are predicted based on the ratings of their similar users; while in Item-based, the new ratings are estimated based on the items similar to those selected by active users. A clear drawback of neighbourhood-based CF is its vulnerability to data sparsity. Particularly, this approach cannot automatically recognize similar indirect neighbours for a given user.
Chapter 4: RW-Based Recommendation Model

Therefore, to alleviate data sparsity issue, some researchers [79–83] adopt RW algorithm for recommendation tasks. In RW, users, items, or both are usually treated as graph nodes, where the edges might serve as ratings or similarities among nodes. It assigns each node a stationary visiting probability as a ranking score and then finds the higher ranked nodes.

In WS recommendation systems, there are two sets of entities, $M$ service users \{u_1, u_2, \ldots, u_m\} and $N$ Web service items \{i_1, i_2, \ldots, i_n\}. Usually, the system record interactions between users and services in a central log. These entities and interactions can be modeled as a graph, in which nodes belong to both user and service sets while edges represent their relations. Each edge has a vector of weighted QoS values (e.g., \(<\text{Response Time (RT)}, \text{Failure Rate (FR)}, \text{Throughput (TH)}, \text{Availability (AV)}, \text{Price (PR)}, \text{etc.}>\)), obtained when user $m$ invokes service $n$, as illustrated in Table 4.1.

<table>
<thead>
<tr>
<th></th>
<th>service(_1)</th>
<th>service(_2)</th>
<th>service(_3)</th>
<th>..</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam</td>
<td>(&lt;\text{RT, FR, TH, AV, PR}&gt;)</td>
<td>(\times)</td>
<td>(&lt;\text{RT, FR, TH, AV, PR}&gt;)</td>
<td>..</td>
</tr>
<tr>
<td>Kevin</td>
<td>(\times)</td>
<td>(&lt;\text{RT, FR, TH, AV, PR}&gt;)</td>
<td>(\times)</td>
<td>..</td>
</tr>
<tr>
<td>Philip</td>
<td>(&lt;\text{RT, FR, TH, AV, PR}&gt;)</td>
<td>..</td>
<td>(&lt;\text{RT, FR, TH, AV, PR}&gt;)</td>
<td>..</td>
</tr>
</tbody>
</table>

### Table 4.1: Sample Recorded QoS Attributes

4.2 Motivation

Although Random Walk algorithm has been successfully used in recommendation models to alleviate data sparsity, it still suffers from poor performance in terms of recommendation accuracy, especially when applied on a classical user-item recommendation model. Taking the advantage of RW into consideration, the main motivation behind this work is to design an effective recommendation model which
utilizes Random Walk algorithm as an underlying approach. Our plan was to find an effective method to better direct the walk using measured similarities among users and services. In fact, an effective Web service recommendation approach needs to infer sound similarities among users/items, rather than to rely solely on direct user ratings to items. By effectively capturing and integrating actual relationships among users and services into our model, recommendation accuracy is improved compared to traditional Random Walk based model.

A key feature of this work is that it takes into account the similarity of Users and Services separately and then integrate them with previously gathered QoS information into a comprehensive recommendation model. By applying Random Walk algorithm on the Integrated Model, higher recommendation accuracy is obtained with greater tolerance to data sparsity.

4.3 Contributions

The key contributions of this chapter are:

1. Presented Jaccard coefficient in several variants that are appropriate for WS recommendation. Given that the transitional Jaccard coefficient effectively works on a binary rating scale, first web service ratings are transformed from n-ary to binary rating scale so that it well fits Jaccard computation.

2. Defined three types of the graph which represent the building blocks of the proposed model. The first is a User-Service Bipartite rating-based graph. The second and third are Unipartite graphs that capture similarities among Users and Services groups respectively.
3. Proposed an effective Integrated-Model QoS-based Graph model for WS recommendation, in which User-Service Bipartite Graph can be fused with User-Based and Service-Based Unipartite Graphs.

4. Applied a Top-K Random Walk recommendation algorithm onto the Integrated-model. The algorithm, after performing a certain number of steps, filters out already user-rated items and then selects the best ones to be recommended to a given user.

5. Conducted a set of extensive experiments on a real-world WS dataset to validate our recommendation model. Comprehensive analysis on the impact of various experimental parameters is also provided. Results show that improved recommendation accuracy can be obtained by utilizing the proposed model in different integration approaches. It also reveals that the model is less prone to data sparsity issue.

The rest of this Chapter is organized as follows. First, section 4.4 introduces the proposed WS recommendation model. Then, section 4.5 presents User-based and Item-based similarity graphs. In section 4.6, given is a description of how the Integrated-Model QoS-based Graph is constructed. The evaluation methodology is then shown in section 4.9. Finally section 4.10 summarizes the chapter.

### 4.4 WS recommendation model

Figure 4.1 illustrates the main stages of the construction of the proposed recommendation model, which are described as follows:

1. Service users contribute their past QoS data to a central log.
Chapter 4: RW-Based Recommendation Model

**Figure 4.1: WS Recommendation Model Construction**

2. User-Service Bipartite graph is created, containing weighted QoS values.

3. Using Jaccard coefficient, similarity magnitudes among *users* are computed to build a User-Based Unipartite graph.

4. Similarity values among *services* are also computed using the same coefficient to create a Service-Based Unipartite graph.

5. User-Service Bipartite graph with User-Based and Service-Based Unipartite graphs are fused to create the IMQG model.
6. Top-k Random Walk algorithm is applied on the IMQG model to generate a list of best $k$ Web services based on their relevance scores, while ignoring services that user already invoked.

This work adopts graph structure as a building block of our model. Thus, it creates three types of graph, i.e., Unipartite, Bipartite, and Integrated-Model graphs, as described in the following sections. Note that the terms rating and weighted QoS value are used interchangeably throughout this paper.

### 4.4.1 User-Service Bipartite Graph

The first graph in the model is User-Service Bipartite Graph, illustrated in Figure 4.2. It is defined as follows:

**Definition 1** (User-Service Bipartite Graph). A User-Service Bipartite Graph is defined as an undirected graph $G = (V, E)$, where $V$ and $E$ represent finite sets of nodes and edges respectively. Set $V$ is a union of two disjoint subsets $V = V_1 \cup V_2$, where $V_1$ consists of user nodes and $V_2$ contains service nodes. Every edge has the form $e = (a, b, w)$ where $a \in V_1$, $b \in V_2$ and $w$ is a weighted QoS value.

Table 4.2 shows a sample graph that is realized as a weighted adjacency matrix $W_{QoS}$, where $S_{1-5}$ are service items and $U_{1-3}$ are service users. The $\times$ cell denotes the absence of the QoS value.

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U_1$</td>
<td>$w_{1,1}$</td>
<td>$w_{1,2}$</td>
<td>$w_{1,3}$</td>
<td>$\times$</td>
<td>$w_{1,5}$</td>
</tr>
<tr>
<td>$U_2$</td>
<td>$\times$</td>
<td>$w_{2,2}$</td>
<td>$\times$</td>
<td>$w_{2,4}$</td>
<td>$w_{2,5}$</td>
</tr>
<tr>
<td>$U_3$</td>
<td>$\times$</td>
<td>$w_{3,2}$</td>
<td>$w_{3,3}$</td>
<td>$w_{3,4}$</td>
<td>$w_{3,5}$</td>
</tr>
</tbody>
</table>
4.5 Similarity Graphs

A key advantage of using graph structure is that its local structural characteristics can be effectively exploited within recommendation techniques. In particular, several similarity forms among graph nodes can be utilized to recommend unknown nodes to active users. This work employs a widely used similarity coefficient (i.e., Jaccard coefficient) to compute similarity magnitudes of every two users/services. The following section first defines similarity graphs and then presents various forms of Jaccard coefficient used in constructing these graphs.
4.5.1 User-Based Unipartite Graph

**Definition 2** (User-Based Unipartite Graph). A User-Based Unipartite Graph is defined as an undirected graph \( G = (V, E) \), where \( V \) and \( E \) represent finite sets of nodes and edges respectively. Set \( V \) consists of user nodes. Every edge has the form \( e = (x_u, y_u, w_{uu}) \) where \( x_u, y_u \in V \) and \( w_{uu} \) is a similarity magnitude between users \( x_u \) and \( y_u \).

4.5.2 Service-Based Unipartite Graph

**Definition 3** (Service-Based Unipartite Graph). A Service-Based Unipartite Graph is defined as an undirected graph \( G = (V, E) \), where \( V \) and \( E \) represent finite sets of nodes and edges respectively. Set \( V \) consists of service nodes. Every edge has the form \( e = (x_s, y_s, w_{ss}) \) where \( x_s, y_s \in V \) and \( w_{ss} \) is a similarity magnitude between services \( x_s \) and \( y_s \).

4.5.3 Jaccard similarity for recommendation

Jaccard similarity coefficient is defined as the size (cardinality) of intersection divided by the size of union of two sets \([84]\). Given that \( E_{u_i} \) and \( F_{u_j} \) are two rating sets of users \( u_i \) and \( u_j \) respectively, Jaccard coefficient can be defined as follows:

\[
J_c(E_{u_i}, F_{u_j}) = \frac{|E_{u_i} \cap F_{u_j}|}{|E_{u_i} \cup F_{u_j}|} \tag{4.1}
\]

Although the definition is straightforward, its implementation can considerably vary resulting in different similarity outcomes. In fact, there are different
approaches to computing the union and intersection of two rating sets based on
the method of handling positive and negative (i.e., liked or disliked) ratings. For
instance, a simple approach evenly treats liked and disliked common items. This is
the most common approach in current Collaborative Recommender Systems. An-
other possible approach is to differentiate between the two types such that greater
weight is assigned to liked items over the disliked ones.

1. $E_{u_1}$ and $F_{u_2}$ are two rating sets of two users $u_1$ and $u_2$ respectively.

2. $p_{i,j}$ is the number of positive ratings in both sets $E_{u_1}$ and $F_{u_2}$.

3. $n_{i,j}$ is the number of negative ratings in both sets $E_{u_1}$ and $F_{u_2}$.

4. $d_{i,j}$ is the number of positive ratings in set $E_{u_1}$ but negative in set $F_{u_2}$ or
   vice versa.

5. $u_{i,j}$ is the number of unrated items in set $E_{u_1}$ but not set $F_{u_2}$ or vice versa.

**Table 4.3: RUNNING EXAMPLE: W_{QoS} WITH BINARY RATING SCALE**

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$\times$</td>
<td>1</td>
</tr>
<tr>
<td>$U_2$</td>
<td>$\times$</td>
<td>0</td>
<td>$\times$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$U_3$</td>
<td>$\times$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Unlike with binary rating scale, illustrated in Table 4.3, computing Jaccard
coefficient is more complicated with n-ary rating scale (e.g., QoS values). Thus, it
first needs to divide the n-ary rating scale into three sets, *good*, *bad* and *unrated*.
Let $G$, $B$ and $Z$ be the sets of *good*, *bad* and *unrated* values, respectively.

$$G = \{x : x > T_{QoS}\}$$

(4.2)
Chapter 4: RW-Based Recommendation Model

\[ B = \{ x : x \leq T_{QoS} \} \quad (4.3) \]
\[ Z = \{ x : x \not\in G \land x \not\in B \} \quad (4.4) \]

where \( T_{QoS} \) is a predefined QoS threshold value. Accordingly, four sets of similarities among a given pair of nodes are defined as:

\[ P_{i,j} = \{ s_k : w_{i,k} \in G \land w_{j,k} \in G, i \neq j \}, \quad p_{i,j} = \{ P_{i,j} \} \quad (4.5) \]
\[ N_{i,j} = \{ s_k : w_{i,k} \in B \land w_{j,k} \in B, i \neq j \}, \quad n_{i,j} = \{ N_{i,j} \} \quad (4.6) \]
\[ D_{i,j} = \{ s_k : (w_{i,k} \in G \land w_{j,k} \in B) \lor \]
\[ (w_{i,k} \in B \land w_{j,k} \in G), i \neq j \}, \quad d_{i,j} = |D_{i,j}| \quad (4.7) \]
\[ U_{i,j} = \{ s_k : (w_{i,k} \in Z \land w_{j,k} \not\in Z) \lor \]
\[ (w_{i,k} \not\in Z \land w_{j,k} \in Z), i \neq j \}, \quad u_{i,j} = |U_{i,j}| \quad (4.8) \]

where \( p_{i,j} \) and \( n_{i,j} \) denote the number of positive and negative ratings, respectively, in both sets \( E_{i,j} \) and \( F_{i,j} \); \( d_{i,j} \) is the number of positive ratings in set \( E_{i,j} \) but negative in set \( F_{i,j} \) or vice versa; and \( u_{i,j} \) is the number of unrated items in set \( E_{i,j} \) but not set \( F_{i,j} \) or vice versa. Based on the above definitions, four different forms to compute Jaccard similarity coefficient are presented, namely, \( Jc_1 \), \( Jc_2 \) and \( Jc_3 \). The simple form of Jaccard coefficient is defined as:

\[
Jc_1(E_{u_i}, F_{u_j}) = \frac{p_{i,j} + d_{i,j} + n_{i,j}}{p_{i,j} + d_{i,j} + n_{i,j} + u_{i,j}}
\]

(4.9)

This definition clearly ignores the quality of rating by uniformly considering
all rated items, whether they are positive or negative. In particular, it computes
the ratio of the number of rated common items to the number of all items except
the completely-unrated common ones. Table 4.4 shows user similarity matrix of
our running example based on \( Jc_1 \).

Table 4.4: \( Jc_1 \) USER SIMILARITY MATRIX

<table>
<thead>
<tr>
<th></th>
<th>( U_1 )</th>
<th>( U_2 )</th>
<th>( U_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U_1 )</td>
<td>1</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>( U_2 )</td>
<td>0.4</td>
<td>1</td>
<td>0.75</td>
</tr>
<tr>
<td>( U_3 )</td>
<td>0.6</td>
<td>0.75</td>
<td>1</td>
</tr>
</tbody>
</table>

As the above definition does not differentiate between positive and negative
ratings, the second form of Jaccard coefficient puts more emphasis on positive
ratings. It is defined as:

\[
Jc_2(E_{u_i}, F_{u_j}) = \frac{p_{i,j}}{p_{i,j} + d_{i,j} + n_{i,j} + u_{i,j}} \quad (4.10)
\]

Obviously, this definition computes the ratio of the number of positively-rated
common items to the number of all items except the completely-unrated common
ones. Table 4.5 shows user similarity matrix of our running example based on \( Jc_2 \).

Table 4.5: \( Jc_2 \) USER SIMILARITY MATRIX

<table>
<thead>
<tr>
<th></th>
<th>( U_1 )</th>
<th>( U_2 )</th>
<th>( U_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U_1 )</td>
<td>1</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>( U_2 )</td>
<td>0.2</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>( U_3 )</td>
<td>0.2</td>
<td>0.25</td>
<td>1</td>
</tr>
</tbody>
</table>

While the previous two definitions take into account negative and partially-
unrated common items to some extent, the last form of Jaccard coefficient entirely
avoids them. It is defined as:

\[
Jc_3(E_{u_i}, F_{u_j}) = \frac{p_{i,j}}{p_{i,j} + d_{i,j}} \quad (4.11)
\]
Table 4.6: Jc₃ USER SIMILARITY MATRIX

<table>
<thead>
<tr>
<th></th>
<th>U₁</th>
<th>U₂</th>
<th>U₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>U₁</td>
<td>1</td>
<td>0.5</td>
<td>0.33</td>
</tr>
<tr>
<td>U₂</td>
<td>0.5</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>U₃</td>
<td>0.33</td>
<td>0.5</td>
<td>1</td>
</tr>
</tbody>
</table>

In this definition, the ratio of the number of positively-rated common items to the number of positively- and differently-rated common ones is computed. Table 4.6 shows user similarity matrix of our running example based on Jc₃.

Although Jc₃ form ignores partially-unrated common items, it essentially focuses on how much positively-rated items exist. However, the last variation of Jaccard coefficient includes also negatively-rated items. In this form, two users are considered similar when their ratings are matched, regardless positively or negatively. The emphasis here is mainly on how similar are the two users in their liked and disliked items. It is defined as:

$$Jc₄(E_{u_i}, F_{u_j}) = \frac{p_{i,j} + n_{i,j}}{p_{i,j} + n_{i,j} + d_{i,j}}$$

Table 4.7: Jc₄ USER SIMILARITY MATRIX

<table>
<thead>
<tr>
<th></th>
<th>U₁</th>
<th>U₂</th>
<th>U₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>U₁</td>
<td>1</td>
<td>0.5</td>
<td>0.33</td>
</tr>
<tr>
<td>U₂</td>
<td>0.5</td>
<td>1</td>
<td>0.66</td>
</tr>
<tr>
<td>U₃</td>
<td>0.33</td>
<td>0.66</td>
<td>1</td>
</tr>
</tbody>
</table>

In this definition, the ratio of the number of positively or negatively-rated common items to the number of positively-, negatively- and differently-rated common ones is calculated. Table 4.7 shows user similarity matrix of our running example based on Jc₄.

From the above definitions, it is evident that measuring similarity among users using Jaccard coefficient is greatly varied depending on the different weights given.
to positive, negative, and unrated items when computing the final similarity magnitude. This flexibility, in addition to its simplicity, adds to the advantages of adopting Jaccard coefficient as a similarity measure in recommendation systems.

This chapter evaluates the four different forms of Jaccard coefficient which are used to construct User-Based and Service-Based similarity graphs. Initially, the expectation was that $J_{c3}$ and $J_{c2}$ will outperform $J_{c1}$ due to the great emphasis on positive rating, which naturally reflects the similarity of users’ tastes.

### 4.6 Integrated-Model QoS-based Graph (IMQG)

Once the User-Service graph and both similarity graphs are created, they are then fused into one graph called the Integrated-Model QoS-based Graph.

**Definition 4** (Integrated-Model QoS-based Graph). An Integrated-Model QoS-based Graph is defined as an undirected graph $G = (V, E)$, where $V$ and $E$ represent finite sets of nodes and edges respectively. Set $V$ is obtained as a union of two subsets $V = V_1 \cup V_2$, where $V_1$ consists of user nodes and $V_2$ contains service nodes. Every edge has the form $e = (x, y, w)$ where $x \in V$ and $y \in V$. If $x \in V_1$ and $y \in V_2$ or vice versa, then $w$ is a weighted QoS value, otherwise it is a similarity value between $x$ and $y$.

As a multi-partite graph, the IMQG model can be viewed as a transition probability matrix. Random Walk over a multi-partite graph is a multi-step approach that simulates a navigating process from one part of the graph to another, e.g., from a set of user nodes to the corresponding set of item nodes. It is a stochastic process in which the initial state is known, while the next state $S$ is provided by a probability distribution. The distribution is represented by constructing the transition probability matrix $A$, where the value of $A_{i,j}$ is the probability of moving
from node $i$ (at time $n$) to node $j$ (at time $n+1$) as in the following:

$$A_{i,j} = P(S_{n+1} = j | S_n = i)$$

(4.13)

Figure 4.3 illustrates how our IMQG model is created as a transition probability matrix. A complete transition matrix is initially created from a number of sub-matrices, namely, User-User, User-Service, Transposition of User-Service and Service-Service sub-matrices. These sub-matrices also have different types of weight. Particularly, User-Service sub-matrix contains weighted QoS values, while User-User and Service-Service sub-matrices store similarity magnitudes. In order to conform to Random Walk algorithm, the next step is to row-normalize the sub-matrices.

If a similarity sub-matrix is excluded from the recommendation process, it is substituted by a self-transition sub-matrix to allow the walk to stay in place.
For example, when applying Random Walk with only service-service similarity sub-matrix, user-user similarity sub-matrix is replaced by user identity matrix, i.e., with ones on the main diagonal and zeros elsewhere. To control the effect of similarity or the self-transition feature, a certain probability $\alpha$ is applied. Additionally, User-Service and its Transposition sub-matrix are row-normalized by the factor $\beta = 1 - \alpha$.

### 4.7 Recommending Web Services using IMQG

This section introduces how Random Walk algorithm is applied to the IMQG recommendation model. First, a generic Top-K Random Walk algorithm is presented, then the main recommender algorithm for the IMQG model is illustrated.

#### 4.7.1 Generic Top-K Random Walk algorithm

The pseudo code of Top-k RW is presented in Algorithm 4.1, in which the initial state query vector $v_0$ indicates an active user. This user is selected to start the walk by setting his corresponding element in $v_0$ to 1, while all remaining elements (i.e., all other users) are initialized to zero:

$A_{n,n}$ is a complete transition probability matrix, i.e., our IQ. The second is an initial state query vector $v_0$, which indicates user(s) selected to start the walk by setting the corresponding elements.

The third variable of the input is $N$, which equals to how many steps are required. Two settings are available, manual and automatic. In manual setting, the system initially determined the value of $N$, while in automatic mode the walk continues until a convergence state is reached. The last parameter $k$ means
Chapter 4: RW-Based Recommendation Model

Algorithm 4.1. Top-k Random Walk over IMQG (TopKRW)

Input:
\( A_{n,n} \), a transition probability matrix (IMQG).
\( v_0 \), an initial state query vector of size \( n \).
\( N \), number of steps.
\( k \), number of top recommendation items required.

Output:
\( v_N \), a final state vector, i.e., \( v_0 \) at step \( N \).

Function:
1: \( t = 1 \)
2: repeat
3: \%Walk one step ahead
4: \( v_0 = v_0 \times A_{n,n} \)
5: \( t = t + 1 \)
6: until \( (t == N) \)
7: Remove already rated\( (v_N) \)
8: sort\( (v_N, \text{desc}') \)
9: \( R = \text{select}(v_N, \text{Topk}) \)
10: return \( R \);

how many recommendations are required from the system. The algorithm then
computes the next state vector by multiplying the previous one by the initial
transition matrix, until the break condition is met. After \( t \) steps, the final state
vector \( v_t \) will contain a new probability distribution over the original graph. After
removing the items already rated by the user, the algorithm ranks the resulted
probabilities which considered final recommendations.

4.7.2 RW-based Recommender Algorithm for IMQG

The recommender algorithm for the proposed IMQG model is given in Algo-
rithm 4.2. The algorithm receives the same set of inputs mentioned in the previous
RW algorithm. In addition, it receives the similarity type variable \( \text{SimType} \) and
Jaccard Coefficient type variable \( \text{JaccType} \). By the former, the integration mode
is determined; while by the latter, the Jaccard coefficient type is specified. The
\text{computeSimilarity} function then executed for \( \text{user} - \text{user} \), \( \text{service} - \text{service} \) or
both, as in (Algorithm 4.2 lines 3), 6, and 9-10 respectively. Once the integration
model is ready, the final IMQG model is created (Algorithm 4.2 line 14), to which the Top – k RW algorithm is then applied (Algorithm 4.2 line 15). Finally, the recommendation results are output.

Algorithm 4.2. Recommender Algorithm of the IMQG model

**Input:**
- $A_{n,n}$, an initial rating matrix (A).
- $v_0$, an initial state query vector of size n.
- $N$, number of steps.
- $k$, number of top recommendation items required.
- $SimType$, similarity type.
- $JaccType$, Jaccard coefficient type.

**Output:**
- $v_N$, a final state vector

**Procedure:**
1: $S_u = S_s = [ ]$
2: if $SimType = \text{'user-user'}$ then
3:  $S_u = \text{ComputeSimilarity}(A, \text{'user-user'}, JaccType)$
4: else
5:  if $SimType = \text{'service-service'}$ then
6:   $S_s = \text{ComputeSimilarity}(A, \text{'service-service'}, JaccType)$
7:  else
8:    if $SimType = \text{'user-service'}$ then
9:     $S_u = \text{ComputeSimilarity}(A, \text{'user-user'}, JaccType)$
10:    $S_s = \text{ComputeSimilarity}(A, \text{'service-service'}, JaccType)$
11:  end if
12: end if
13: end if
14: $\bar{A} = \text{CreateIMQGs}(A, S_u, S_s)$
15: $R = \text{TopKRW}(\bar{A}, v_0, N, k)$;
16: Display recommendations in $R$
4.8 Running Example

In this section, a detailed example \(^1\) of how the Model is created for a simple recommendation task, is presented. The initial assumption is that the invocation records of 3 users of 5 web services containing QoS observations (Response Time in considered here) are available. The input is a rating 3 × 5 matrix in which average QoS of services to all users are provided as in matrix \(A_{3,5}\):

\[
A_{3,5} = \begin{bmatrix}
320 & ? & 290 & 530 & 610 \\
? & 870 & 658 & 100 & 960 \\
501 & 1000 & 550 & 410 & 500
\end{bmatrix}
\]

A question mark entry \((A_{i,j} = ?)\) indicates that the corresponding user \(u_i\) has no knowledge about the QoS of service \(s_j\), i.e., has never invoked it. Hence, the role of a recommender system is to assist \(u_i\) in selecting one of these unknown services to him according to some preferences. To execute a recommendation task, three main steps are required, namely, computing similarity model(s), constructing IMQG model, and finally applying Top-k Random Walk algorithm.

1. **Computing similarity model(s)** First, it transforms the rating matrix from n-ary rating scale to binary one. This is achieved by thresholding the former scale using a predefined value such as \(\text{median}(QoS_{RT})\), i.e., \(\mu_{RT} = \)

\(^1\) A complete example is in Appendices
530. Consequently, the good, bad, and unrated sets are as follows:

\[ G = \{(u_1, s_5), (u_2, s_2), (u_3, s_2), (u_3, s_3), (u_4, s_2), (u_4, s_3)\} \]

\[ B = \{(u_1, s_1), (u_1, s_3), (u_1, s_4), (u_2, s_1), (u_3, s_4), (u_4, s_4), (u_4, s_5)\} \]

\[ Z = \{(u_1, s_2), (u_2, s_3), (u_2, s_4), (u_2, s_5), (u_3, s_1)\} \]

Now the number of similar/dissimilar abstracted ratings for each pair of users, i.e., positive-positive, negative-negative, differing, rated-unrated counters is computed. As a result, six sets of counters where each of which belongs to a specific pair of users, namely, user_1 vs user_2, user_1 vs user_3, user_1 vs user_4, user_2 vs user_3, user_2 vs user_4, and user_3 vs user_4 are computed as follows:

<table>
<thead>
<tr>
<th>user_1 vs. user_2</th>
<th>user_1 vs. user_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_{1,2} = {} )</td>
<td>( P_{1,3} = {(s_5)} )</td>
</tr>
<tr>
<td>( p_{1,2} = 0 )</td>
<td>( p_{1,3} = 1 )</td>
</tr>
<tr>
<td>( N_{1,2} = {(s_1)} )</td>
<td>( N_{1,3} = {(s_4)} )</td>
</tr>
<tr>
<td>( n_{1,2} = 1 )</td>
<td>( n_{1,3} = 1 )</td>
</tr>
<tr>
<td>( D_{1,2} = {} )</td>
<td>( D_{1,3} = {(s_3)} )</td>
</tr>
<tr>
<td>( d_{1,2} = 0 )</td>
<td>( d_{1,3} = 1 )</td>
</tr>
<tr>
<td>( U_{1,2} = {(s_2), (s_3), (s_4), (s_5)} )</td>
<td>( U_{1,3} = {(s_1), (s_2)} )</td>
</tr>
<tr>
<td>( u_{1,2} = 4 )</td>
<td>( u_{1,3} = 2 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>user_1 vs. user_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_{1,3} = {} )</td>
</tr>
<tr>
<td>( N_{1,3} = {(s_1), (s_4)} )</td>
</tr>
<tr>
<td>( D_{1,3} = {(s_3), (s_5)} )</td>
</tr>
<tr>
<td>( U_{1,3} = {(s_2)} )</td>
</tr>
</tbody>
</table>
Using the above sets, a full $4 \times 4$ user-user Jaccard-based similarity matrix, $Sim_{4,4}^{J_{\text{user}}}$ is created. For illustration purposes, the second form of Jaccard coefficient is used, $Jc_2(E_u, F_u) = p_{i,j} / (p_{i,j} + d_{i,j} + n_{i,j} + u_{i,j})$.

\[
Sim_{4,4}^{J_{\text{user}}} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1
\end{bmatrix}
\]

\[
Sim_{3,3}^{J_{\text{user}}} = \begin{bmatrix}
1.00 & 0.00 & 0.20 & 0.00 \\
0.00 & 1.00 & 0.20 & 0.20 \\
0.20 & 0.20 & 1.00 & 0.40 \\
0.00 & 0.20 & 0.40 & 1.00
\end{bmatrix}
\]
Since the similarity matrix must be symmetric, only the upper part is calculated and then it is replicated to fill the matrix. Interestingly, based on similarity concept, a user must be exactly similar to himself, i.e., \( Sim_{u_2,u_2} = 1 \).

However, applying \( Jc_2 \) to a user against himself can produce other than 1 (e.g., \( Jc_{1,2} = \frac{1}{1+0+1+3} = 0.25 \)).

Likewise, similar/dissimilar abstracted ratings for each pair of services, i.e., positive-positive, negative-negative, differing, rated-unrated counters are calculated. Thus, 10 sets of counters are obtained where each of which belongs to a specific pair of services.

<table>
<thead>
<tr>
<th>service(_1) vs. service(_2)</th>
<th>service(_1) vs. service(_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_{1,2} = { } ), ( p_{1,2} = 0 )</td>
<td>( P_{1,3} = { } ), ( p_{1,3} = 0 )</td>
</tr>
<tr>
<td>( N_{1,2} = { } ), ( n_{1,2} = 0 )</td>
<td>( N_{1,3} = {(u_1)}, \ n_{1,3} = 1 )</td>
</tr>
<tr>
<td>( D_{1,2} = {(u_2),(u_4)}, \ d_{1,2} = 2 )</td>
<td>( D_{1,3} = {(u_4)}, \ d_{1,3} = 1 )</td>
</tr>
<tr>
<td>( U_{1,2} = {(u_1),(u_3)}, \ u_{1,2} = 2 )</td>
<td>( U_{1,3} = {(u_2),(u_3)}, \ u_{1,3} = 2 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>service(_1) vs. service(_4)</th>
<th>service(_1) vs. service(_5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_{1,4} = { } ), ( p_{1,4} = 0 )</td>
<td>( P_{1,5} = { } ), ( p_{1,5} = 0 )</td>
</tr>
<tr>
<td>( N_{1,4} = {(u_1),(u_4)}, \ n_{1,4} = 2 )</td>
<td>( N_{1,5} = {(u_4)}, \ n_{1,5} = 1 )</td>
</tr>
<tr>
<td>( D_{1,4} = { } ), ( d_{1,4} = 0 )</td>
<td>( D_{1,5} = {(u_1)}, \ d_{1,5} = 1 )</td>
</tr>
<tr>
<td>( U_{1,4} = {(u_2),(u_3)}, \ u_{1,4} = 2 )</td>
<td>( U_{1,5} = {(u_2),(u_3)}, \ u_{1,5} = 2 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>service(_2) vs. service(_3)</th>
<th>service(_2) vs. service(_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_{2,3} = {(u_3),(u_4)}, \ p_{2,3} = 2 )</td>
<td>( P_{2,4} = { } ), ( p_{2,4} = 0 )</td>
</tr>
<tr>
<td>( N_{2,3} = { } ), ( n_{2,3} = 0 )</td>
<td>( N_{2,4} = { } ), ( n_{2,4} = 0 )</td>
</tr>
<tr>
<td>( D_{2,3} = { } ), ( d_{2,3} = 0 )</td>
<td>( D_{2,4} = {(u_3),(u_4)}, \ d_{2,4} = 2 )</td>
</tr>
<tr>
<td>( U_{2,3} = {(u_1),(u_2)}, \ u_{2,3} = 2 )</td>
<td>( U_{2,4} = {(u_1),(u_2)}, \ u_{2,4} = 2 )</td>
</tr>
</tbody>
</table>
Using the above sets, a full \(5 \times 5\) service-service Jaccard-based similarity matrix, \(Sim_{service}^{Jc_{5,5}}\), is then created. The second form of Jaccard coefficient is also used.

\[
Sim_{service}^{Jc_{5,5}} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & \frac{0}{0+0+2+2} & \frac{0}{0+1+1+2} & \frac{0}{0+2+0+2} & \frac{0}{0+1+1+2} \\
0 & \frac{2}{2+0+0+2} & \frac{2}{2+0+0+2} & \frac{0}{0+0+2+2} & \frac{1}{1+0+1+2} \\
0 & \frac{2}{2+0+0+2} & \frac{2}{2+0+0+2} & \frac{0}{0+0+2+2} & \frac{1}{1+0+1+2} \\
0 & \frac{0}{0+1+1+2} & \frac{0}{0+1+1+2} & \frac{0}{0+1+2+0} & \frac{0}{0+1+2+0} \\
0 & \frac{1}{1+0+1+2} & \frac{1}{1+0+1+2} & \frac{0}{0+1+2+0} & \frac{1}{1+0+1+2}
\end{bmatrix}
\]
Chapter 4: RW-Based Recommendation Model

\[ Sim^{Jc_{service}}_{5,5} = \begin{bmatrix}
1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 1.00 & 0.50 & 0.00 & 0.25 \\
0.00 & 0.50 & 1.00 & 0.00 & 0.33 \\
0.00 & 0.00 & 0.00 & 1.00 & 0.00 \\
0.00 & 0.25 & 0.33 & 0.00 & 1.00
\end{bmatrix} \]

2. Constructing IMQG model Having both similarity matrices; the Integrated Model can be constructed by combining them with the original rating matrix and its transposition as follows:

\[
IMQG = \begin{bmatrix}
\alpha \cdot Sim^{Jc_{service}}_{3,3} & (1 - \alpha) \cdot A \\
(1 - \alpha) \cdot A^T & \alpha \cdot Sim^{Jc_{service}}_{5,5}
\end{bmatrix}
\]

\[
= \begin{bmatrix}
1.00 & 0.00 & 0.20 & 0.00 & 320 & 0 & 290 & 530 & 610 \\
0.00 & 1.00 & 0.20 & 0.20 & 420 & 550 & 0 & 0 & 0 \\
0.20 & 0.20 & 1.00 & 0.40 & 0 & 870 & 658 & 100 & 960 \\
0.00 & 0.20 & 0.40 & 1.00 & 501 & 1000 & 550 & 410 & 500 \\
320 & 420 & 0 & 501 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0 & 550 & 870 & 1000 & 0.00 & 1.00 & 0.50 & 0.00 & 0.25 \\
290 & 0 & 658 & 550 & 0.00 & 0.50 & 1.00 & 0.00 & 0.33 \\
530 & 0 & 100 & 410 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \\
610 & 0 & 960 & 500 & 0.00 & 0.25 & 0.33 & 0.00 & 1.00
\end{bmatrix}
\]

After row-normalizing the original rating matrix along with its transposition and then multiplying by \(\alpha = 0.2\), the following ready-for-recommendation matrix is finally obtained.

79
Chapter 4: RW-Based Recommendation Model

3. Applying Top-k Random Walk algorithm

Vector $v_0$ is assigned the initial values for corresponding users and services (i.e., the first 4 entries is determined to users, while the remaining 5 entries are dedicated for services). To recommend to a particular user, for instance user$_2$, user$_2$ is selected by setting its corresponding entry to 1 ($v_0[user_2] = 1$) and then apply Random Walk algorithm in odd iteration number such as 3 times. Each simulate the transition from user set to service set and vice versa. Moreover, since user$_2$ is not familiar with service$_2$, service$_3$ and service$_5$ (no rating), the recommender is required to predict a probability value for both services to the user.

While $v_0$ is the input zero vector, except at index 2, $\bar{v}_0$, $\bar{\bar{v}}_0$ and $\bar{\bar{\bar{v}}}_0$ are its subsequent versions at step 1, 2 and 3 respectively.

The final step is to extract the new stationary probability of all initially unknown services to user$_2$ and order them ascendingly and then inform him. Thus the list is ($service_4 = 0.058$, $service_5 = 0.069$ and $service_3 = 0.082$ respectively). Result show that $service_4$ is the highly recommended service to user$_2$. 

80
Chapter 4: RW-Based Recommendation Model

$$A_{9,9} = \begin{bmatrix}
0.200 & 0.000 & 0.000 & 0.000 & 0.146 & 0.000 & 0.133 & 0.242 & 0.279 \\
0.000 & 0.200 & 0.000 & 0.000 & 0.346 & 0.454 & 0.000 & 0.000 & 0.000 \\
0.000 & 0.000 & 0.200 & 0.000 & 0.000 & 0.269 & 0.203 & 0.031 & 0.297 \\
0.000 & 0.000 & 0.000 & 0.200 & 0.135 & 0.270 & 0.149 & 0.111 & 0.135 \\
0.206 & 0.271 & 0.000 & 0.323 & 0.200 & 0.000 & 0.000 & 0.000 & 0.000 \\
0.000 & 0.182 & 0.288 & 0.331 & 0.000 & 0.200 & 0.000 & 0.000 & 0.000 \\
0.155 & 0.000 & 0.351 & 0.294 & 0.000 & 0.000 & 0.200 & 0.000 & 0.000 \\
0.408 & 0.000 & 0.077 & 0.315 & 0.000 & 0.000 & 0.000 & 0.200 & 0.000 \\
0.236 & 0.000 & 0.371 & 0.193 & 0.000 & 0.000 & 0.000 & 0.000 & 0.200
\end{bmatrix}$$

$$v_0 = \begin{bmatrix}
0.00 & 1 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00
\end{bmatrix}$$

$$\bar{v}_0 = \begin{bmatrix}
0.000 & 0.200 & 0.040 & 0.040 & 0.271 & 0.182 & 0.000 & 0.000 & 0.000
\end{bmatrix}$$

$$\bar{v}_0 = \begin{bmatrix}
0.041 & 0.219 & 0.068 & 0.105 & 0.121 & 0.097 & 0.044 & 0.016 & 0.032
\end{bmatrix}$$

$$\bar{v}_0 = \begin{bmatrix}
0.047 & 0.137 & 0.077 & 0.091 & 0.126 & 0.120 & 0.082 & 0.058 & 0.069
\end{bmatrix}$$

4.9 Evaluation and Experiments

This section presents the evaluation process of the proposed recommendation model. In the following subsections, we discuss the dataset used, the testing methodology, recommendation accuracy results, and finally the impact of some experimental parameters on the recommendation quality.
Chapter 4: RW-Based Recommendation Model

4.9.1 Dataset

To evaluate the effectiveness of our recommendation model, a series of experiments based on a real QoS dataset, provided by Zibin Zheng [77] is conducted. The dataset is distributed in 150 files, each of which represents a service user’s invocation record. In each file, there are about 10,000 Web service invocations on 100 publicly available Web services. In total, the dataset contains more than 1.5 million invocations.

First, two $150 \times 100$ matrices are constructed, namely, Response Time and Failure Rate matrices. Given that service user $u$ has invoked service item $i$ about 100 times, the approach calculates Response Time rating as the mean Response Time of all invocations, Table 4.8 shows part of RTT matrix. On the other hand, Failure Rate rating is computed as the ratio of a number of failures to the total number of invocations. One is added to shift all zero entries to be conformed with RW approach, Table 4.9 shows part of FR matrix. Finally, both ratings are assigned to the corresponding entries in RT and FR matrices respectively. Both QoS values are defined as follows:

$$RT_{i}^{avg} = \frac{\sum_{j=1}^{n} invokations_{ij}^{RT}}{n}, \ldots, n = |invokations_{i}| \approx 100$$

$$FR_{i} = \frac{|F|}{n}, \ldots, n = |invokations_{i}| \approx 100$$

$$F = \{ f : f \in invokations_{i, status}^{status} \land f \neq \text{OK} \}$$

where $invokations_{i}$ is the set of invocation records for service $i$ by a given user. Each record $j$ contains response time $invokations_{ij}^{RT}$ measurement and returning status $invokations_{i,j}^{status}$ along with other information. Set $F$ is the set of all invocations of service $i$ that have a returning status other than 'OK', indicating the failure of execution.
4.9.2 Testing Methodology

We adopt a testing methodology similar to the one described in [85]. For each QoS attribute, i.e., RT and FR, the set of service users $U$ is split into two subsets: a training set $U_N$ and test set $U_T$. As a result, four sub-matrices are created, namely, $RT_{U_N}$, $RT_{U_T}$, $FR_{U_N}$ and $FR_{U_T}$. To simulate sparse real-world data, some entries (i.e., QoS values) are randomly remove from training matrices with different densities (i.e., 10%, 20%, etc.). Similarly, the density of test matrices is also lowered to 50% to provide a necessary set of unrated service items required for next evaluation process.

For each test user, a different number of entries is also removed. The number of remaining entries is called a \textit{Given Number} which represents the number of QoS values provided by a test user. The removed entries are the top-rated ones to ensure that their corresponding Web services are reasonably relevant to the respective test user. All removed entries for all test users are stored in set $T$. 

\begin{table}[h]
\centering
\caption{RTT MATRIX}
\begin{tabular}{|c|c|c|c|}
\hline
 & $sw_1$ & $sw_2$ & $sw_{100}$ \\
\hline
$u_1$ & 80 & 3846 & ... & 4562 \\
$u_2$ & 586 & 458 & ... & 64 \\
$u_3$ & 61 & 720 & ... & 1520 \\
... & ... & ... & ... & ... \\
$u_{150}$ & 245 & 92 & ... & 43 \\
\hline
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\caption{FR MATRIX}
\begin{tabular}{|c|c|c|c|}
\hline
 & $sw_1$ & $sw_2$ & $sw_{100}$ \\
\hline
$u_1$ & 1 & 1 & ... & 1 \\
$u_2$ & 1.0092 & 1.0102 & ... & 1 \\
$u_3$ & 1 & 1.0170 & ... & 1 \\
... & ... & ... & ... & ... \\
$u_{150}$ & 1.0095 & 1.0107 & ... & 1.0020 \\
\hline
\end{tabular}
\end{table}
Chapter 4: RW-Based Recommendation Model

Finally, original values of the removed entries are used to estimate prediction accuracy.

To measure the prediction accuracy of the proposed Top-k recommendation approach, \( \text{recall}@k \) and \( \text{precision}@k \) metrics are used. For each test service item \( i \) of test user \( u \):

1. The approach randomly selects 30% of service population (i.e., 33 items of the used dataset), unrated by user \( u \), assuming that most of them are not of interest to him.

2. Then it predicts the ratings for test service item \( i \) and the additional selected service items.

3. A ranked list by ordering all the 34 service items based on their predicted ratings is then formed. Let \( q \) denote the rank of the test service item \( i \) within the list. The best result is where \( i \) precedes all other items, i.e., \( q = 1 \).

4. A top-k recommendation list is constructed by selecting the \( k \) top-ranked service items from the list. If \( q \leq k \), then it is a hit (i.e., the test service item \( i \) is recommended to user \( u \)), otherwise it is a miss. The probability of hit increases with \( k \). When \( k = 34 \), there is always a hit.

Intuitively, recall of a test service item can be either 0 (in the case of miss) or 1 (in the case of hit). Similarly, precision can be either 0 or \( \frac{1}{k} \). Therefore, the overall \( \text{recall}@k \) and \( \text{precision}@k \) are defined by averaging over all test cases:

\[
\text{recall}(k) = \frac{\# \text{hit}}{|T|}
\]
Chapter 4: RW-Based Recommendation Model

\[ \text{precision}(k) = \frac{\#\text{hit}}{|T| \cdot k} \]

where \(|T|\) is the total number of test entries of all test users and \(k\) is the given size of recommendation list.

To study the effectiveness of incorporating similarity computation into WS recommendation, three different recommendation approaches are presented, namely, RW with User-Based similarity, RW with Service-Based similarity and RW with User-Service-Based similarity approaches. Similarity of each approach is applied using four forms of Jaccard coefficient resulting into twelve recommendation approaches (i.e., \(\text{RWUUJc}_1, \text{RWSSJc}_1, \text{RWUSJc}_1, \text{RWUUJc}_2, \text{RWSSJc}_2, \text{RWUSJc}_2, \text{RWUUJc}_3, \text{RWSSJc}_3, \text{RWUSJc}_3, \text{RWUUJc}_4, \text{RWSSJc}_4, \text{RWUSJc}_4\)), as illustrated in Table 4.10.

Since Response Time and Failure Rate are both classified as negative QoS attributes (i.e., the lower the attribute value is, the better the quality is), the approach carefully takes that into account in two cases. The first is when it picks Top-\(k\) Web services from a whole recommendation list (i.e., at the last step of the Top-\(k\) RW algorithm); while the second is when it splits the n-ary QoS scale into positive and negative subscales in order to compute Jaccard coefficient. For comparison, the basic Random Walk is adopted as a baseline approach to show to what extent the proposed IMQG model outperforms RW in terms of recommendation accuracy and sparsity tolerance. It also empirically sets \(T_{QoS} = \text{median}(QoS_x)\), number of steps of RW \(N = 3\) and \(\alpha = 0.2\) in all experiments.
Table 4.10: Evaluation Approaches

<table>
<thead>
<tr>
<th>Approach ID</th>
<th>Random-Walk based</th>
<th>User-User Similarity</th>
<th>Service-Service Similarity</th>
<th>Jaccard1 Coefficient</th>
<th>Jaccard2 Coefficient</th>
<th>Jaccard3 Coefficient</th>
<th>Jaccard4 Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUUJc1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWSSJc1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUSJc1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUUJc2</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWSSJc2</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUSJc2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUUJc3</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWSSJc3</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUSJc3</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUUJc4</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWSSJc4</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUSJc4</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.9.3 Recommendation Accuracy

Table 4.11 and Table 4.12 show recall results for Response Time and Failure Rate attributes respectively. Different recommendation approaches are examined, employing 10%, 20% and 30% density of RT and FR training matrices. While the training user number is set to 100, given numbers are varied from 10 to 30 (i.e., $G_{10}$, $G_{20}$ and $G_{30}$). Top-K is also fixed at 10. Each experiment is executed 50 times and mean recall values are reported. Experimental results of Table 4.11 and Table 4.12 show that:

1. Recommendation approaches based on $Jc_3$ and $Jc_2$ outperform $Jc_4$ based approach, while $Jc_1$ comes at the end of the list. In fact, $Jc_1$ performs
as poorly as basic Random Walk. The reason is that \( J_{C_1} \) uniformly treats positive and negative ratings, while \( J_{C_3} \) and \( J_{C_2} \) mainly focus on positive ratings. Therefore, designing effective similarity measures is key to accurate WS recommendation. However, \( J_{C_3} \) based approach is the winner in most cases.

2. Service-based and User-Service-based integrated approaches (i.e., \( RWSSJ_{C_x} \) and \( RWUSJ_{C_x} \)) mostly produce better accuracy results than User-based approaches (i.e., \( RWUUJ_{C_x} \)). In fact, this result indicates that item-based similarity approaches have higher accuracy performance than user-based ones. However, more investigation on larger datasets is required to confirm this result.

3. Competition between Service-based and User-Service-based approaches is evident. However, the former tends to perform slightly better in low densities and given numbers.

4. With the increase of training matrices density, the recommendation accuracy is greatly improved, since denser training matrices offer more information for producing high-quality recommendations.

5. By increasing given number from 10 to 30, recall values are enhanced. This observation indicates that users can receive more accurate recommendations when there is more QoS information provided by service users.
Table 4.11: Accuracy Performance Comparison (Response Time)

<table>
<thead>
<tr>
<th>Methods</th>
<th>Training Users = 100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Recall</td>
</tr>
<tr>
<td></td>
<td>Density = 10%</td>
</tr>
<tr>
<td></td>
<td>G10</td>
</tr>
<tr>
<td>RW</td>
<td>28.16%</td>
</tr>
<tr>
<td>RWUUJc1</td>
<td>28.12%</td>
</tr>
<tr>
<td>RWSSJc1</td>
<td>29.20%</td>
</tr>
<tr>
<td>RWUSJc1</td>
<td>29.63%</td>
</tr>
<tr>
<td>RWUUJc2</td>
<td>28.91%</td>
</tr>
<tr>
<td>RWSSJc2</td>
<td>32.82%</td>
</tr>
<tr>
<td>RWUSJc2</td>
<td>34.14%</td>
</tr>
<tr>
<td>RWUUJc3</td>
<td>32.27%</td>
</tr>
<tr>
<td>RWSSJc3</td>
<td>35.72%</td>
</tr>
<tr>
<td>RWUSJc3</td>
<td>35.64%</td>
</tr>
<tr>
<td>RWUUJc4</td>
<td>33.16%</td>
</tr>
<tr>
<td>RWSSJc4</td>
<td>35.15%</td>
</tr>
<tr>
<td>RWUSJc4</td>
<td>32.39%</td>
</tr>
</tbody>
</table>
### Table 4.12: Accuracy Performance Comparison (Failure Rate)

<table>
<thead>
<tr>
<th>Methods</th>
<th>Training Users = 100</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Density = 10%</td>
<td>G10</td>
<td>G20</td>
<td>G30</td>
<td>G10</td>
<td>G20</td>
<td>G30</td>
<td>G10</td>
<td>G20</td>
<td>G30</td>
<td>G10</td>
</tr>
<tr>
<td>RW</td>
<td>27.70%</td>
<td>27.98%</td>
<td>28.20%</td>
<td>27.06%</td>
<td>25.20%</td>
<td>23.97%</td>
<td>27.13%</td>
<td>27.02%</td>
<td>25.20%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUUJc1</td>
<td>29.51%</td>
<td>29.73%</td>
<td>28.62%</td>
<td>27.74%</td>
<td>26.56%</td>
<td>30.34%</td>
<td>29.51%</td>
<td>29.01%</td>
<td>24.46%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWSSJc1</td>
<td>29.05%</td>
<td>28.33%</td>
<td>27.38%</td>
<td>28.02%</td>
<td>32.90%</td>
<td>29.33%</td>
<td>28.36%</td>
<td>28.70%</td>
<td>32.10%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUSJc1</td>
<td>27.75%</td>
<td>28.42%</td>
<td>28.16%</td>
<td>29.63%</td>
<td>31.06%</td>
<td>28.88%</td>
<td>30.58%</td>
<td>28.68%</td>
<td>29.85%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUUJc2</td>
<td>29.28%</td>
<td>28.26%</td>
<td>27.48%</td>
<td>31.70%</td>
<td>28.85%</td>
<td>31.35%</td>
<td>31.79%</td>
<td>35.50%</td>
<td>31.93%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWSSJc2</td>
<td>35.79%</td>
<td>37.92%</td>
<td>35.28%</td>
<td>35.93%</td>
<td>37.49%</td>
<td>39.71%</td>
<td>36.16%</td>
<td>39.61%</td>
<td>40.75%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUSJc2</td>
<td>34.07%</td>
<td>35.21%</td>
<td>36.72%</td>
<td>36.23%</td>
<td>38.32%</td>
<td>39.28%</td>
<td>36.26%</td>
<td>39.02%</td>
<td>39.41%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUUJc3</td>
<td>34.90%</td>
<td>32.74%</td>
<td>34.11%</td>
<td>34.06%</td>
<td>31.80%</td>
<td>33.72%</td>
<td>33.65%</td>
<td>34.58%</td>
<td>32.20%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWSSJc3</td>
<td>36.23%</td>
<td>37.90%</td>
<td>38.31%</td>
<td>35.54%</td>
<td>39.19%</td>
<td>41.07%</td>
<td>37.45%</td>
<td>40.74%</td>
<td>39.90%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUSJc3</td>
<td>35.05%</td>
<td>37.01%</td>
<td>39.51%</td>
<td>35.75%</td>
<td>37.86%</td>
<td>42.15%</td>
<td>37.15%</td>
<td>39.83%</td>
<td>38.18%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUUJc4</td>
<td>29.58%</td>
<td>28.52%</td>
<td>28.50%</td>
<td>32.22%</td>
<td>31.29%</td>
<td>30.20%</td>
<td>31.49%</td>
<td>29.61%</td>
<td>28.12%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWSSJc4</td>
<td>30.67%</td>
<td>13.56%</td>
<td>12.30%</td>
<td>30.42%</td>
<td>11.57%</td>
<td>10.42%</td>
<td>31.00%</td>
<td>12.16%</td>
<td>10.00%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWUSJc4</td>
<td>19.53%</td>
<td>13.16%</td>
<td>11.06%</td>
<td>13.51%</td>
<td>11.42%</td>
<td>10.95%</td>
<td>14.19%</td>
<td>10.09%</td>
<td>10.06%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.9.4 Impact of Experimental Parameters

Intuitively, the outcomes of recommendation tasks are usually influenced by a set of factors such as how many options the user is looking for, how familiar the user is with the system, and how knowledgeable the system itself is, etc. These can be interpreted as Top-k, Given Number and Matrix Density parameters respectively. Hence, it is important that the evaluation of recommendation models considers such factors. In the following, the impact of different experimental factors (also called parameters) on accuracy performance is studied. Experiments are varied in terms of settings such as the underlining recommendation approaches, QoS attributes, evaluation metrics and experimental parameters as in the following:

4.9.4.1 Top-K

This parameter determines the cardinality of recommendation set that the user receives (i.e., the number of options he may be interested in exploring). Consequently, effective recommender model is the one which can retrieve the highest $k$ items of user’s interest.

This experiment examines the impact of Top-k parameter on the accuracy of the Service-based recommendation approach. It varies the value of Top-k from 2 to 20 with a step value of 2. Figure 4.4 and 4.5 report recall and precision values of RT matrix under the experimental settings of training matrix density = 10%, given number = 20, and training user number = 100. Based on the same settings, Figure 4.6 and 4.7 show recall and precision results of FR matrix.

- **Response Time**

  Figure 4.4 shows that the recall results consistently increase with Top-k
value. This observation is justified by the fact that by increasing Top-k parameter, the possibility of a test item to hit is higher. This positively contributes to the overall recall@k value. Consequently, Top-k parameter has a positive influence on the accuracy performance. In addition, recall results of RWSSJc3 and RWSSJc2 steadily outperform the results of RWSSJc1 and RW. RWSSJc4 also comes in the middle in both set of results. This observation is connected to the fact that Jc3 and Jc2 heavily consider positive ratings. Therefore, selecting similarity among users focusing on their positive ratings is more effective. While equally treating positive and negative ratings lead only to poor similarity results and thereby to low recommendation accuracy. However, in most real-world recommender systems, this parameter is delimited by 20 since users usually do not look beyond 20 items in recommendation lists. From this experiment, the conclusion is that integrating Service similarities based on Jaccard3 along the original user-service rating
data achieves best recommendation accuracy.

Moreover, figure 4.5 shows that the precision results mostly decrease with Top-k value. This observation is similarly reasoned by the fact that by increasing Top-k parameter, the possibility of a test item to hit is higher. However, the increase in the number of hits is not in line with increase in Top-k leading to an overall decrease trend. Obviously, this is a reflection to the form of this metric. Consequently, Top-k parameter has a negative impact on the accuracy performance. In addition, the order of approaches is still the same where RWSSJc3 is at the top, and RW at the bottom. This observation adds another evidence that considering positive ratings is favoured on equally treating them with negative ratings.

- **Failure Rate**

  Figure 4.6 shows that the recall results also consistently increase with Top-k
value. This observation again is another confirmation of the positive contribution to the overall recall@k value. Regarding the order of the competing approaches, it is noticeable that there is a slight difference where RWSSJc4 has replaced RW at the bottom of the list. Recall that RWSSJc4 focuses on exact similarity between users neglecting their quality. This repeatedly indicates that taking common negative ratings into account when generating Jaccard coefficient has a negative effect on the overall accuracy. This is reasonable since negatively rated items are obviously of no interest to the user which distracts the RS. Also, RWSSJc1 comes in the middle in both sets of results. Moreover, Figure 4.7 shows that the precision results mostly steady with increasing Top-k value. This observation is reasoned by the fact that the corresponding recall results have a narrower range in this experiment. That makes the increase in the number of hits is almost in line with increase in Top-k leading to an steady overall trend with the exception of RWSSJc4.
In addition, the order of approaches is still the same where RWSSJc3 is at the top, and RWSSJc4 at the bottom. Therefore, discriminating negatively rated items is recommended for an effective recommendation model.

By utilizing precision in the above experiments, it is shown that this metric does not accurately reflect the recommendation accuracy. Therefore, in the next experiments, the emphasis is on recall as a principal evaluation metric while studying the impact of other parameters.

4.9.4.2 Given Number

This factor specifies user’s degree of familiarity with the recommender system, i.e., to what extent the user has already actively interacted with the system. In particular, this is realized by given number of user’s ratings which are already
recorded within the recommender knowledge, before the new recommendation query of the user. To this end, the initial expectation is that the more ratings the user already has within the system, the better recommendation he can receive. Hence, in these four experiments, the goal is to analyse the impact of the given number on recommendation accuracy. First, it changes the given number from 5 to 50 with a step value of 5. Then it sets other experimental parameters as follows: training user number = 100 and 130, Top-k = 10, training matrix density = 10% and 20%. Jaccard3 based approaches is used, as it has performed the best in the previous experiments, while all integrations modes are compared.
Figure 4.8: Comparison of Given Number
Figures 4.8(a) and 4.8(b) report recall results of RT matrix with density training matrix = 10\%, while figures 4.9(a) and 4.9(b) employ density training matrix = 20\%. The experimental results of Figure 4.8 and 4.9 show that the recommendation accuracy of the RW\textsubscript{SSJc3} and RW\textsubscript{USJc3} approaches are significantly enhanced with large given numbers compared with RW\textsubscript{USJc3} and RW. This observation indicates that by providing more Web service QoS ratings, the service user can obtain an enhanced recommendation list of unknown web services. Regarding the order of performance, the superiority of Service-based and User-Service based integration modes is still witnessed. The other observation is that basic RW has almost no sensitivity to higher given numbers compared with similarity-based approaches. This observation provides additional evidence that the proposed similarity-based RW approach performs better with active user involvement. This idea encourages users to actively contribute in delivering their QoS data, in order to receive better recommendations.
Chapter 4: RW-Based Recommendation Model

Figure 4.9: Comparison of Given Number
4.9.4.3 Training Matrix Density

This parameter determines overall level of knowledge-ability of the recommender system, in regards to all users. This is expressed by the density of ratings the system already has. It is directly related to data sparsity, i.e., the more the system has dense data, the less the sparsity issue has an effect. Therefore, the following experiment is dedicated to studying the influence of the training matrix density on recommendation accuracy. The settings are as follows: it first varies the density from 5% to 50% with a step value of 5%. Then it sets: training user number = 100, Top-k = 10 and 15, given number = 10 and 20. Figure 4.10 shows the recall results of RT matrix, where Figure 4.10 (a) and 4.10(b) employ given number of 10, while 4.11 (a) and 4.11 (b) employ given number of 20.

The experimental results of Figure 4.10 and 4.11 indicate that, although the recall trend is not as good as the one in the previous experiments, the accuracy performance, in general, is improved when the density increases from the low 5% upward. However, the enhancement acceleration becomes slower with large density values. This observation implies that enhanced recommendation accuracy can be obtained by collecting more QoS data, which also prompts the end user to contribute his QoS recordings for better recommendations. The other interesting observation is that RWSSJc3 and RWUSJc3 approaches mostly surpass the rest approaches especially at low densities (i.e., a case of sparse data), which in turn proves that these approaches can effectively handle sparse data and still give the desired accuracy. Finally, inserting similarity information that is effectively measured through Jaccard coefficient eliminates the defectiveness of basic RW recommendation approach in handling sparse rating data.
Figure 4.10: Comparison of Training Matrix Density
Chapter 4: RW-Based Recommendation Model

Figure 4.11: Comparison of Training Matrix Density
This chapter answered the four questions, mentioned at the beginning.

The first question is, *What type of knowledge is required?* The knowledge required in the proposed model is the typical knowledge in recommender systems (i.e., users, services, and ratings). In particular, the model is designed as a generic one where different types of ratings can be utilized, including QoS statistics, explicit user ratings, or implicit usage data. However, the presentation and evaluation of the model achieved using QoS properties and a public QoS dataset respectively.

The second question is *How to represent the knowledge?* As this thesis adopts the Graph-based approach, all entities participated in this model are identified as components of the graph, i.e., nodes and edges and their weights. Moreover, multiple types of graphs are employed for recording these components such as Unipartite, Bipartite, and Integrated graphs.

The third question is, *How to deal with the lack of recommendation data?* It handles the sparsity problem, i.e., the lack of recommendation, by utilizing the Random Walk algorithm as the base on which the proposed model is built on. However, for its low performance, the approach captures and integrates more relationships between users and services into our model to better direct the walk using measured similarities among users and services.

The final question is, *How to recommend appropriate individual services to end user?* The proposed model is capable of recommending high ranked services to end user as in the following: first, it constructs User-Service Bipartite Graph that contains weighted QoS attributes representing the relations between Users
and services they have previously invoked. **Second**, Jaccard similarity coefficient is used to construct similarity Unipartite Graphs among users and services separately. **Third**, by fusing all the three graphs in one model, it creates the Integrated-Model QoS-based Graph, on which a Top-k random walk approach is applied. **Finally**, a list of high ranked Web services is recommended to end users.

To evaluate our model, comprehensive experiments are conducted on a real-world QoS dataset where the results show that our WS recommendation approach achieves significant improvement in recommendation accuracy and effective tolerance to data sparsity.
Chapter 5

Enhanced Similarity based model for web service recommendation

In chapter 4, we have discussed our recommendation model for web services in details. Although the validation methodology revealed the effectiveness of integrating Jaccard-based similarities along with the ratings, the applicability of the model is concerned due to the size of rating data in real-world application domains. As a result, the next step is an attempt to improve the model in terms of applicability. In this chapter, our goal is to reduce the model before applying Random Walk algorithm, in a step that will accordingly reduce the computation time. However, maintaining recommendation accuracy is a must that should be considered.

The chapter starts with explaining the reduction process and then its validation. To test the applicability of the model as a generic recommendation model, a further evaluation process in conducted using additional dataset with a variety of evaluation metrics and common recommendation approaches. This chapter answers the following research questions:
Chapter 5: Enhanced Similarity Based Model

1. How efficient is the proposed model?

2. Can the model be applied to other recommendation domains?

5.1 Introduction

In real-world recommendation systems, there is usually a huge number of users, compared to a relatively smaller number of items. For example, the number of movies in a network like Netflix is less than 100000, while the number of its subscribers is currently over 60 million. The same realization applies to the case of Web services, where millions of Internet users work with a limited number of Web services. In this context, finding an applicable (i.e., optimized) recommendation approach to deal with such numbers, while maintaining quality effectiveness of the recommendation outcomes, becomes a necessity, from a system designer’s perspective. This chapter presents a proposed approach to optimize the original recommendation model presented earlier in this paper.

5.2 Contributions

The following are the key contributions of this chapter:

- Introduced an optimized recommendation model, in which RW algorithm is better guided using a selected subset of Jaccard similarities, instead of the entire similarity set. The goal is to minimize complexity of the proposed model, based on conventional RW algorithm, while maintaining a acceptable level of recommendation accuracy.
Proposed a new measure to study how much the proposed model is reduced compared to the original domain. This assists in observing the behavior of the model in response to different experimental settings.

Conducted a set of extensive experiments on a real-world WS dataset to validate our recommendation framework. Comprehensive analysis of the impact of various experimental parameters is also provided.

Further evaluation of the proposed model as a generic recommendation model in terms of data domain. A movie rating dataset along with a publicly available recommendation library are used in the evaluation process. In addition, several common recommendation approaches are compared with using various evaluation metrics. Results show the effectiveness of the model as a generic recommendation model that can be applied in different recommendation application.

The rest of this Chapter is organized as follows. First, Section 5.3 introduces the proposed reduced WS recommendation model. Then, Section 5.4 presents the updated recommender algorithm. In section 5.5, a description of the evaluation of the reduced model is demonstrated. Section 5.6 verifies the generality of the proposed model while the summary of the chapter is presented in section 5.7.

5.3 Enhanced Recommendation Model

The proposal of the enhanced recommendation model is an attempt to reduce the recommendation domain on an individual basis, before starting the recommendation process. In other words, it first transforms the recommendation model form a complete/unified to a reduced/personalized model on which the RW-based
algorithm is then applied. Model reduction is accomplished through selecting a limited set of users who are marked as the closest users to current user $u_i$. Consequently, a selection measure is needed, by which the selection of the minimized set of users is specified. From a system designer’s point of view, the entire similarity domain (i.e., including all users), is considered a similarity search space in which each couple of users are connected using an edge. The edge is weighted with the pre-computed similarity magnitude between the two users. The role of the selection measure here is to estimate how far a user is within this space to the current user. To perform that, a Similarity Selection Threshold ($SST$) is used by which less-similar users are excluded and only the closer ones are chosen. the resulted set of closely similar users to user $u_i$ is called as Reduced Similarity Set ($RSS_{u_i}$). Similarity Selection Threshold is defined as follows:

**Definition 5** (Similarity Selection Threshold ($SST$)): Similarity Selection Threshold is a decimal value between [0-1], where a user $u_j$ is considered similar to a given user $u_i$ if their similarity magnitude is equal or greater than $SST$ value. As a result, $u_j \in RSS_{u_i}$ which represents the reduced recommendation model for user $u_i$.

**Definition 6** (Reduced Similarity Set ($RSS_{u_i}$)): Reduced Similarity Set is a subset of a set of all users where each of its element has a similarity value equal or greater than the given $SST$ value with current user.

$$RSS_{u_i} = \{u_j : S_{u_i, u_j} \geq SST\} \tag{5.1}$$

where $S_{u_i, u_j}$ is the similarity magnitude between $u_i$ and $u_j$.

As the proposed recommendation model adopts Jaccard coefficient as the similarity measure, the value of $SST$ is the least Jaccard similarity magnitude at
which two users are considered closely similar to each other. For example, if $SST = 0.9$, then all users who have similarity magnitudes between $[0.9-1.0]$ with the given user will belong to the reduced set of users $RSS_u$. In other words, when computing recommendations to $u_i$, not all users with their ratings will be involved, rather only the ones who are in his reduced similarity set. As a result, applying the final recommendation approach (i.e., RW algorithm) on the reduced set will consume less computation time. Aiming at faster processing, it is also required that the final recommendation results are of a close quality to the results when applying a complete-domain/non-enhanced recommendation approach. Figure 5.1 illustrates a sample recommendation model, in which several users are placed at different distances from current user. The distance represents the similarity magnitude, while the circles represent various values of $SST$. The figure shows how selecting $SST$ value (e.g., $SST = 0.9$) produces a smaller similarity domain.

In fact, the enhanced recommendation approach can be successfully realized as follows. The Reduced Set of Similar users for each user can be stored in his
profile, so that every time recommendations are required, the RSS is ready and need not be created again. This achieves a great reduction in computation time. However, there must be an update strategy in which this set is recomputed again either periodically or after a certain number of updates operations recorded on the entire set. As the ratio of the number of new items to the number of existing ones is practically low, such an update strategy can be effectively applied. For instance, in E-commerce, users tend to do much search before making a real purchase. Additionally, many Recommender systems currently follow the push approach in which the recommendations will come up automatically once users navigate to an online page. Therefore, the total number of updates to the Recommender knowledge base is significantly less than the number of readings.

5.4 Recommender Algorithm of Reduced IMQG

This section presents the recommender algorithm for reduced IMQG model (Algorithm 5.1). The input of the algorithm includes the same set of inputs mentioned in the algorithm 4.2, section 4.7.2. Note that, the computeSimilarity function is abstracted (Algorithm 5.1 line 1). The next line refers to CreateIMQG function which is responsible to build the entire similarity-based IMQG model including optional user – user, service – service or both (Algorithm 5.1 line 2). The filtering process is applied at (Algorithm 5.1 lines 3-9), where the ratings of every user who has a similarity value less than the threshold $\theta$ are excluded. Once the reduced model is prepared, the Top-k RW algorithm is executed (Algorithm 5.1 line 10) and then the final recommendation results are displayed (Algorithm 5.1 line 11).
Chapter 5: Enhanced Similarity Based Model

Algorithm 5.1. Recommender Algorithm of the Reduced IMQG

Input:
- $A_{n,n}$, an initial rating matrix ($A$).
- $v_0$, an initial state query vector of size $n$.
- $N$, number of steps.
- $k$, number of top recommendation items required.
- $SimType$, similarity type.
- $JaccType$, Jaccard coefficient type.

Output:
- $v_N$, a final state vector

Procedure:
1: $[S_u, S_s] = \text{ComputeSimilarity}(A, SimType, JaccType)$
2: $\bar{A} = \text{CreateIMQGs}(A, S_u, S_s)$
3: for each $u_i$ node in $Users$ do
4: \hspace{1em} if $u_i \neq user$ then
5: \hspace{2em} if $Sim(user, u_i) < \theta$ then
6: \hspace{3em} Exclude $u_i$ rating knowledge from $\bar{A}$;
7: \hspace{1em} end if
8: \hspace{1em} end if
9: \hspace{1em} end for
10: $R = \text{ExecuteRW}(\bar{A}, v_0, N, k)$;
11: Display recommendations in $R$
5.5 Evaluation of the Reduced IMQG

In this section, a series of experiments is conducted to ensure the effectiveness of the enhanced recommendation approach in terms of quality of outcomes and amount of speedup (i.e., computation time reduction). In addition, a key point is to investigate to what extent the reduction of recommendation model affects the ability of RW algorithm to deal with data sparsity issue. In other words, the goal is to have the proposed approach maintain the ability to cope with the common sparsity problem. In the following experiments, the $SST$ is employed to tune the enhanced recommendation model. By varying the value of $SST$, the size of $RSS$ is controlled which in turn affects the recommendation results. Note that whenever $SST$ is set to 0.0, that means the non-enhanced recommendation approach is applied (i.e., all similarity values are included). On the other hand, if $SST$ is set to 1.0, this means that only completely matching users are considered.

5.5.1 Recommendation Accuracy

To show the effectiveness of the proposed enhanced model in terms of recommendation accuracy, the following series of experiments varies the settings to study Domain Reduction Ratio ($DRR$), impact of Top-K parameter, and impact of Jaccard coefficient.

5.5.1.1 Domain Reduction Ratio (DRR)

The purpose of this experiment is to study the effect of applying the enhanced recommendation model with various values of $SST$ (i.e., different sizes of Reduced
Chapter 5: Enhanced Similarity Based Model

Similarity Set), in terms of recommendation accuracy and Domain Reduction Ratio. Domain reduction ratio is the measure that computes how much the domain is reduced compared to the entire domain and is defined as follows:

$$DRR_{ui} = \frac{U_N - \text{sizeof}(RSS_{ui})}{U_N}$$  \hspace{1cm} (5.2)

where $U_N$ is the total number of users in the domain.

The settings of the experiment are as follows: $SST$ is varied from 0.99 to 0.00, given number = 20, training user number = 100, Top-k = 10, training matrix density = 20%. In addition, recommendation approach used is RWUSJc3. Figure 5.2 shows the results of the experiment.
Chapter 5: Enhanced Similarity Based Model

Results show that there is no significant impact of DRR values over the recommendation accuracy. In other words, when DRR is high (i.e., only very similar users to given user are considered), the recall results were also high. For instance, when SST = 0.99, only users who have similarity magnitude greater or equal to 0.99 with current user \( u_i \) will be included in his Reduced Similarity Set \( RSS_{u_i} \), which is expected to be a small number. Subsequently, DDR value will be high which also means that the rating domain is significantly reduced. From the figure, it is obvious that while the DRR value decline (following SST decline), the average Recall results continue within a small range. Two main observations can be recorded. The first is that the model mainly recognizes the active members of the domain when recommending services to current users, i.e., discriminating inactive members. The second observation is that the system is highly tolerant to domain reduction, which means adaptability to sparse problem. Finally, results of this experiment confirm that the enhanced recommendation approach has a positive performance in producing accurate recommendations with reduced recommendation domain and hence effectively minimizing computation time.

5.5.1.2 Impact of Top-K

In this experiment, the impact of Top-K parameter over the optimized recommendation model is studied with various SST values. In terms of recommendation accuracy, both recall and precision are measured against a range of Top-K values.

The settings of the experiment are as follows: Top-K is varied from 2 to 20 with a step value of 2, given number = 20, training user number = 100, training matrix density = 20% . The used recommendation approach is RWSSJc3. Figure 5.3 and 5.4 report recall and precision results respectively.
Results indicate that Top-K parameter still has a positive impact on recommendation accuracy of the optimized model. Similar to prior experiments, there is no negative effect on accuracy by reducing the recommendation domain. All competing optimized approaches have close recall results which also provides another evidence to the effectiveness of our proposed enhanced model.

### 5.5.1.3 Impact of Jaccard Coefficient

The goal of this experiment is to study the impact of various Jaccard similarity forms over the enhanced recommendation model using multiple SST values. The experiment employs recall and precision to test the recommendation accuracy.
Chapter 5: Enhanced Similarity Based Model

The settings of the experiment are as follows: Top-K is set to 15, given number = 20, training user number = 100, training matrix density = 30%. The used recommendation approaches are RWUSJc1, RWUSJc2, RWUSJc3 and RWUSJc4. Figure 5.5 and 5.6 report recall and precision results respectively.

Results demonstrate that RWUSJc3 and RWUSJc2 approaches steadily outperform both RWUSJc4 and RWUDJc1. Similar to non-enhanced recommendation approach, RWSSJc4 comes in the middle in both sub-experiments. The other observation is that the proposed model maintains consistent recommendation accuracy across the range of SST values, acknowledging the success employment of the proposed recommendation model.
5.5.2 Computation Time

To show the effectiveness of the proposed enhanced model in terms of reducing computation time, the following series of experiments is conducted with various settings by varying Given Number, Density and Top-K parameters. In all experiments, the recommendation approach used is RWSSJc with the four different types of Jaccard coefficient, within Optimized/Enhanced (OP) and Non-optimized/normal (N) recommendation models. Therefore, the result is eight different recommendation approaches.
Figure 5.6: Comparison of SST vs. Precision-Optimized-Jaccard(s)

5.5.2.1 Given Number

This experiment is designed to calculate the time (in seconds) required to generate a list of recommendations to end user while varying the corresponding given number. The settings are: *given number* is varied from 5 to 40 with a step value of 5, *training user number* = 100, *Top-k* = 10, *training matrix density* = 20%

Figure 5.7 reports *physical clock Time* results using RT matrix with all Jaccard-based approaches, each of which in normal and optimized versions. The experimental results of Figure 5.7 show that the computation time of all the optimized versions outperforms their non-optimized counterparts at all times. The obvious justification is that the reduction of recommendation model from all users to a set of limited but highly similar ones leaves lower number of ratings, i.e., less
computation time to process them. The second interesting observation is that \textit{RWSSJc1OP} and \textit{RWUSJc2OP} outperform the remaining. Knowing that, unlike Jc1, Jc2 has a good reputation in terms of accuracy; the result indicates that their initial Jaccard calculation put less emphasis on separating ratings according to their quality, which resulted in low similarity values. As a result, the \textit{SST} filtering process excludes most of the missed similarity values leaving only a small RSS set, which obviously interprets the faster performance. It is also noticeable that the trend of all is decreasing with larger number of Given Number. This is justified by the fact that the number of test entries will decrease with large given numbers which in turn results in fewer comparisons to produce the final results.

5.5.2.2 Density

The goal of this experiment is to measure the computation time required to process recommendation tasks with different \textit{training matrix density} values. The settings
are: *training matrix density* is varied from 0.05 to 0.5 with a step value of 0.05, 
given number = 20 training user number = 100, Top-k = 10.

Figure 5.8 shows *Time* results using RT matrix with all Jaccard-based approaches, each of which in normal and optimized versions. Results are similar to the previous experiment in terms of superiority of the optimized approaches over their non-optimized counterparts. However, the *RWSSJc3OP*, which is the optimized service-service approach using Jaccard 3, outperforms the remaining approaches in time consumption reduction, excluding the exceptional case of *RWSSJc1OP* and *RWUSJc2OP*. It is also evident that the trend of all is steady within a certain range. This can be justified by the fact that increasing density has no significant effect on the number of users excluded. This observation proofs that the proposed optimized recommendation model performs well enough in low densities. Therefore, it can be considered a successful sparsity-tolerance model.
Chapter 5: Enhanced Similarity Based Model

5.5.2.3 Top-K

The goal of this experiment is to measure the computation time required to process recommendation tasks with different Top-K values. The settings are: Top-k is varied from 2 to 20 with a step value of 2, given number = 20 training user number = 100, training matrix density = 20%.

Figure 5.9 reports Time results using RT matrix with all Jaccard-based approaches, each of which in normal and optimized/Enhanced versions. The experimental results of Figure 5.9 confirm the result of the above two experiments that the optimized approach outperforms its non-optimized counterpart at all times. However, it is observed that the distinction is larger now between RWSSJc1OP and RWUSJc2OP and the rest. Furthermore, there is no clear trend of all approaches. This can be interpreted by the fact that increasing Top-K has no role in determining the number of users selected in RSS. This observation indicates that
using higher Top-K values has no extra cost of time consumption, while producing better recall results. Finally, this also adds to the features of the proposed optimized recommendation model.

5.6 Generality of the Proposed Model

The previous experiments evaluated the proposed model using a real-world QoS dataset. However, the model is initially designed as a generic recommendation model in which a variety of recommendation tasks with different recommendation domains can be achieved. For instance, a typical rating domain (i.e., using 1-5 scale) for items other than web services can also be applied to the proposed model. Particularly, the graphs of the model are set to the rating data first which are then normalized during the initialization process of the Random Walk. Therefore, to verify the generic nature of the model, in the following section, a dataset called FilmTrust [86] is used to further study the behaviour of the proposed model with non-QoS domains.

5.6.1 Evaluation Environment

FilmTrust is a movie 5-star rating dataset that is crawled from the entire FilmTrust website in June, 2011. It contains a total number of ratings of 35497 rated by 1508 users to 2071 movies. Also the dataset is a sound instance of a sparse data model with sparsity rate of 1.14%, which makes it eligible to verify the performance of recommender algorithms in real-world applications. The dataset is commonly used to evaluate recommendation algorithms.
Moreover, a common open-source recommendation library called Librec\(^1\) is utilized in the evaluation. Librec is a Java Library for Recommender Systems that implements a suit of state-of-the-art and up-to-date recommendation algorithms. Specifically, it contains three major components: Generic Interfaces, Data Structures and Recommendation Algorithms. We implemented and added our model to the available recommendation algorithms.

In the following, a list of brief descriptions of the additional evaluation metrics specifically used in next experiments is presented. Given that \( k \) is the maximum number of items that can be recommended and \( N \) is the cardinality of the set of users:

1. **Normalized discounted cumulative gain (NDCG):** this metric measures the performance of a recommendation system based on the graded relevance of the recommended items. Accordingly, it varies from 0.0 to 1.0, where 1.0 represents the ideal ranking of the items. This metric is commonly used in information retrieval and to evaluate recommender systems.

\[
DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i + 1)} \quad (5.3)
\]

\[
nDCG_k = \frac{DCG_k}{IDCG_k} \quad (5.4)
\]

where \( IDCG_k \) is the maximum possible (ideal) DCG for a given set of items.

2. **Mean average precision (MAP):** is a measure that provides a single-figure measure of quality across recall levels. The MAP for a set of users is the

\(^1\)http://www.librec.net
mean of the average precision scores for each user.

\[
AvgP_k = \sum_{i=1}^{n} \frac{\text{Precision}(i)}{k}
\]  

(5.5)

\[
MAP_k = \frac{\sum_{j=1}^{N} AvgP_{kj}}{N}
\]  

(5.6)

3. **Mean Reciprocal Rank (MRR)** MRR is a statistic measure for evaluating a list of responses to given requests, ordered by probability of correctness. The reciprocal rank is the multiplicative inverse of the rank of the first correct answer. The mean reciprocal rank is the average of the reciprocal ranks of the results of a whole set of requests Q.

\[
MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}
\]  

(5.7)

where \( \text{rank}_i \) refers to the rank position of the first relevant item for the \( i-th \) request.

4. **Area under the ROC curve (AUC)** is a commonly used measure mainly for characterizing the trade-off between true positives and false positives as a threshold parameter (i.e., the number of items predicted as relevant) is varied.

### 5.6.2 Evaluating The Proposed Model

This experiment is designed to study the behaviour of the model in terms of the level of user interaction with recommender system. With such a sparse dataset, the initial expectation is that the number of ratings the user provides is really
crucial to the quality of recommendation results. Hence, the impact of the given number on recommendation accuracy is analysed. The settings of this experiment are as follows: First, the given number is varied from 5 to 25 with a step value of 5. All other settings are complied with the implementation standard of Librec. The Jaccard-based Random Walk approach is used with item-item similarity, i.e., RWSSJc3.

Specifically, the training user number in fact equals to the total number of users. However, some users are considered test users if they have a number of ratings larger than the Given Number parameter. In other words, the splitting of training and testing users is performed based on determining test entries rather than test users. This means that Given Number parameter and the number of test entries (i.e., also indicates the number of test users) are negatively correlated. For instance, providing a small number of given entries results in a high number of test entries. This method is different from the initial approach of evaluation, followed in the previously conducted experiments. Furthermore, the Top-k parameter is set to 5 while the training matrix density is the test matrix density subtracted from the total rating matrix density (1.14%).

Figure 5.10 demonstrates the result of the experiment where Figure 5.10 (a) presents precision@5, precision@10, recall@5 and recall@10 result, while Figure 5.10 (b) shows AUC, MAP, NDCG and MRR result respectively.

Clearly, recommendation accuracy improves positively with increasing the number of given items per user, i.e., Given Number. This is justified by the fact that the more the user contributes to the recommendation model, the better the system can infer direct and indirect relations among similar users and items. This has an ultimate beneficial impact on the quality of final recommendations. Also, this confirms the need to encourage users to effectively interact with the
system and continuously provide their feedbacks and ratings in order to enhance system ultimate outcomes.

5.6.3 Evaluating The Reduced Model

While the previous experiment studied the behaviour of the proposed model, in this experiment, the performance of the reduced model with FilmTrust dataset is analysed. In particular, the reduction threshold is varied while the quality of outcome is observed in terms of the significance of reduction over accuracy. Similarly, with considerable sparsity, the expectation was that the overall performance downgrades with increasing the reduction threshold, i.e., lowering the density of the training set.

The settings of this experiment are as follows: First, the reduction threshold is varied through \{0.1, 0.25, 0.50, 0.75, 0.90\}. All other settings are complied with the implementation standard of Librec. The experiment also uses the Jaccard-based Random Walk approach using item-item similarity, i.e., RWSSJc3. Similar to previous experiment, some users are considered test users if they have a number of ratings larger than the Given Number parameter, where the Given Number is set to 5. Figure 5.11 demonstrates the result of the experiment where Figure 5.11 (a) presents precision@5, precision@10, recall@5 and recall@10 result, while Figure 5.11 (b) shows AUC, MAP, NDCG and MRR result respectively.

Figure 5.11 shows that recommendation accuracy significantly decreases with increasing the reduction threshold from 0.1 to 0.9. This is reasonable due to the fact that the higher the reduction threshold is the lower the training density becomes, resulting in fewer user ratings available. Consequently, the system has no sufficient information about user’s preferences/selections which considerably decreases its ability to discover desired connections between the user and new items.
Chapter 5: Enhanced Similarity Based Model

(A) Precision and Recall vs. Given Number

(B) Ranking Metrics vs. Given Number

Figure 5.10: The Proposed Model Using FilmTrust
Chapter 5: Enhanced Similarity Based Model

(A) Precision and Recall of the Reduced Model

(B) Ranking Metrics of the Reduced Model

Figure 5.11: The Reduced Model Using FilmTrust
Although performing the reduction positively impact the complexity, the result suggests that picking the correct reduction threshold must be carefully applied by the system administrator with close observation to recommendation accuracy. Finally, the result also indicates the positive impact of user contributions on recommendation outcomes.
5.6.4 Comparison With Other Recommendation Approaches

In order to verify the performance of the proposed model, a comprehensive experiment is conducted where twelve commonly used recommendation approaches from different categories are compared with our recommendation model. These approaches belong to different recommender categories such as Collaborative Filtering and Dimensionality Reduction. They also vary in terms of ranking technique, i.e., rating ranking and item ranking approaches. As the proposed model is classified under the item ranking approaches, where the recommendation list contains an ordered items without actual ratings, the comparison with item ranking is straightforward. However, to compare with rating ranking approaches, these approaches are first configured to produce ratings upon which the final recommendation list (i.e., item ranking list) is then generated. This way enables us to conduct the comparison with a wide range of spectrum of recommendation approaches and gives us an idea about where the proposed approach is placed within this spectrum. The Librec recommendation library is also used in this experiment where the metrics studied are precision@5, precision@10, recall@5, recall@10, AUC, MAP, NDCG and MRR.

The settings of this experiment are as follows: the standard settings of the twelve approaches are used as according to initial configuration of Librec. For our proposed model, Jaccard-based Random Walk approach is used with item-item similarity, i.e., RWSSJc3. Test users are the ones who have a number of ratings larger than the Given Number parameter, where the Given Number is set to 5.

Figure 5.12 demonstrates the result of the experiment where each row contains the eight metric magnitudes for a particular recommendation approach. The last row presents the result of the RWSSJc3 approach. In general, RWSSJc3 performed very well, occupying the first place of half of the metrics and the second
place of a couple of them. Specifically, RWSSJc3 surpasses all competents in precision@5, precision@10, AUC and MRR while comes second in recall@5 and recall@10. However, in MAP and NDCG, it failed to compete for the top places. The worse result is in NDCG which is mainly depends of the graded relevance of the recommended items. This indicates that the approach discriminates the order of the recommendation items due to its stochastic nature. However, the result assures that enriching the base Random Walk recommendation model with similarity information among items considerably enhances the performance of the Random Walk, as a generic recommendation model, in comparison to widely used recommendation models.
**Figure 5.12: Comparisons with the Reduced Model (FilmTrust)**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Precision@5</th>
<th>Precision@10</th>
<th>Recall@5</th>
<th>Recall@10</th>
<th>AUC</th>
<th>MAP</th>
<th>NDCG</th>
<th>MRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserCluster</td>
<td>34.69%</td>
<td>31.37%</td>
<td>10.77%</td>
<td>18.37%</td>
<td>84.36%</td>
<td>23.19%</td>
<td>58.91%</td>
<td>69.65%</td>
</tr>
<tr>
<td>ItemCluster</td>
<td>2.41%</td>
<td>1.85%</td>
<td>0.65%</td>
<td>0.95%</td>
<td>39.23%</td>
<td>3.37%</td>
<td>33.06%</td>
<td>7.28%</td>
</tr>
<tr>
<td>UserAvg</td>
<td>29.39%</td>
<td>29.03%</td>
<td>9.70%</td>
<td>17.36%</td>
<td>84.19%</td>
<td>21.69%</td>
<td>57.31%</td>
<td>65.17%</td>
</tr>
<tr>
<td>ItemAvg</td>
<td>0.21%</td>
<td>0.23%</td>
<td>0.05%</td>
<td>0.09%</td>
<td>39.63%</td>
<td>2.77%</td>
<td>31.10%</td>
<td>1.12%</td>
</tr>
<tr>
<td>GlobalAvg</td>
<td>32.31%</td>
<td>29.80%</td>
<td>10.06%</td>
<td>17.27%</td>
<td>83.57%</td>
<td>21.23%</td>
<td>56.99%</td>
<td>65.16%</td>
</tr>
<tr>
<td>SlopOne</td>
<td>1.12%</td>
<td>1.16%</td>
<td>0.25%</td>
<td>0.50%</td>
<td>50.15%</td>
<td>6.07%</td>
<td>35.59%</td>
<td>4.00%</td>
</tr>
<tr>
<td>ItemKNN</td>
<td>47.77%</td>
<td>46.79%</td>
<td>12.26%</td>
<td>23.72%</td>
<td>93.13%</td>
<td>48.12%</td>
<td>69.33%</td>
<td>62.45%</td>
</tr>
<tr>
<td>UserKNN</td>
<td>57.77%</td>
<td>52.84%</td>
<td>20.32%</td>
<td>33.24%</td>
<td>89.82%</td>
<td>45.60%</td>
<td>72.73%</td>
<td>77.17%</td>
</tr>
<tr>
<td>RankALS</td>
<td>50.93%</td>
<td>46.50%</td>
<td>14.10%</td>
<td>23.30%</td>
<td>51.40%</td>
<td>32.05%</td>
<td>60.66%</td>
<td>68.21%</td>
</tr>
<tr>
<td>WRMF</td>
<td>56.41%</td>
<td>52.47%</td>
<td>18.51%</td>
<td>31.19%</td>
<td>74.66%</td>
<td>45.22%</td>
<td>70.13%</td>
<td>73.56%</td>
</tr>
<tr>
<td>SLIM</td>
<td>47.78%</td>
<td>38.94%</td>
<td>16.97%</td>
<td>25.06%</td>
<td>85.13%</td>
<td>29.63%</td>
<td>64.29%</td>
<td>73.33%</td>
</tr>
<tr>
<td>FISMrmse</td>
<td>5.31%</td>
<td>4.64%</td>
<td>1.22%</td>
<td>2.11%</td>
<td>48.75%</td>
<td>4.56%</td>
<td>36.19%</td>
<td>15.47%</td>
</tr>
<tr>
<td>RWSSJc3</td>
<td><strong>57.90%</strong></td>
<td><strong>53.32%</strong></td>
<td><strong>18.59%</strong></td>
<td><strong>31.25%</strong></td>
<td><strong>94.00%</strong></td>
<td><strong>22.10%</strong></td>
<td><strong>36.82%</strong></td>
<td><strong>74.15%</strong></td>
</tr>
</tbody>
</table>
5.7 Summary

This chapter answered the two questions, mentioned at the beginning.

The first question is, *How efficient is the proposed model?* Due to the fact that the complexity of an algorithm greatly impacts its applicability, the work in this chapter attempted to improve the applicability of the proposed model by targeting the reduction of its complexity. Therefore, the proposed model is reduced by introducing the use of a similarity selection threshold by which the recommendation space excludes unlikely supportive users. As a result, the RW algorithm is better guided using a promising subset of Jaccard similarities, instead of the entire similarity set. To maintain an acceptable level of recommendation accuracy, a new measure is proposed to study how much the proposed model is reduced compared to the original domain. Time complexity and recommendation approach are both studied through a series of experiments.

The second question answered is, *Can the model be applied to other recommendation domains?* To test whether the proposed model is applicable to other recommendation domains such as common 5-star rating systems, a further evaluation is conducted using a movie rating dataset within a publicly available recommendation library where several common recommendation approaches are compared with using various evaluation metrics.

Results show the effectiveness of the model as a generic recommendation model that can be applied in different recommendation applications.
Chapter 6

Agent-Based Model to Web Service Composition

6.1 Introduction

In Chapter 4 and 5, a model for recommending individual web services has been proposed. In this chapter, the focus is on designing an effective approach to recommend a composite web service when no individual service is available, which is also called Web Service Composition (WSC). Most of current approaches are categorized as centralized WSC approaches. However, such traditional approaches suffer from critical problems such as performance bottleneck and single point of failure. On the other hand, Agent technology has proved its significance in coping with decentralized problems. Due to the evident analogy between Service and Agent Computing paradigms, the incorporation of both technologies will likely lead to a more powerful hybrid service model. This chapter answers the following questions:
1. How to recommend composite web services in a complexity-balanced manner?

2. How effective is the proposed composition approach?

6.2 Agent and Service Technologies

Software agents work in a distributed environment, in which their interoperability and collaboration provide an effective approach to decompose a complicated task into several simpler subtasks\[66\]\[87\]. A similar paradigm applies to web services by which diverse organizations offer their services to the whole community to facilitate their collaboration. This analogy suggests that incorporating both technologies will likely lead to a more productive model than applying each one individually. The majority of current web service composition approaches are categorized as centralized approaches and thereby they are prone to critical problems such as performance bottleneck and single point of failure. In addition, as web services are naturally distributed across geographical boundaries and are constantly updated, an equivalent level of distribution should accordingly be provided. In other words, software agents are effective means to represent web services for the following reasons [66]:

1. Web services are inactive software components that lack the ability to find and communicate with others. In contrast, Agents are active entities that can search, communicate and perform requested tasks. For example, agents can perform QoS assessments and choose preferred solutions/services among available alternatives.
2. Web services are self-describing components with limited internal knowledge, while agents’ knowledge include surrounding world information which they usually utilize in reasoning and communicating with their peers.

3. Using Web services requires that requesters first learn how to call them by analyzing their published profiles, which is definitely a non-trivial skill. In contrast, agents can automatically find a potential service based on user’s request.

### 6.3 The Contributions

The contributions of this chapter are summarized in the following:

1. It introduces a distributed service dependency model based on AND/OR graph structure. The model effectively encodes the dependency relationships among services based on the analogy with the producer-consumer generic model. The model is decomposed and distributed among individual members of the Agent community such that an agent may participate in composition tasks.

2. Within the agent, the knowledge is stored in a simplified AND/OR model. AND nodes are assigned to service agents while OR nodes to data parameters those agents consume or produce. A reasoning approach is also proposed which is designed to find quality sub-composition solutions in terms of applicability and communication cost. Through exploiting well-established connections within the model, the approach generates and favours the applicable composite solution with the least communication cost.
3. It presents an Agent Communication Protocol for service composition, derived from ACL standard. The protocol contains a short list of messages that agents can use to coordinate their mutual efforts in fulfilling a composition task requested. Upon receiving certain message, the agent utilizes the protocol to respond, either immediately or after consulting other agents, with its final decision to participate or not.

4. It evaluates the proposed composition model using a public composition dataset and results indicate the effectiveness of the approach in terms of reducing communication cost needed to respond to given composition requests.

Some researchers proposed solutions that decompose the overall service dependency model into submodels. For instance, Tong et al. [66] propose an agent-based service composition model in which service and agent technologies are integrated into one cohesive model. The authors present a BDI agent model which contains: belief, action, and plan components, alongside a communication protocol. However, service relations are not considered in the final composition workflows, besides the lack of a dynamic inter-reasoning process within the service agent. In fact, our work is initially inspired by this work. Liang and Su [88] formalize a service composition as a search problem over an AND/OR graph. They introduce a search algorithm to determine the satisfaction of a set composite service requirements. The work is based on the dependencies over functional properties, i.e., input/output parameters, in a centralized composition approach. Wang et al. [89] present an automated Web service composition method supporting conditional branch structures. The authors in [87] propose a new concept called service intention and service extension, based on which they formulate a constraint-aware service composition technique.
Chapter 6: Agent-Based Model To WSC

To the best of our knowledge, none of the existing WSC approaches presents a comprehensive model that adopt a well designed distributed SDG scattered among cooperating agent community, through using an ACL-like communication protocol. In addition, the effective design of the inter-reasoning process of the agent AND/OR graph based knowledge enables us to apply and propose further enhancements to select preferred services to end users dynamically.

The rest of the chapter is organized as follows. Section 6.4 gives essential definitions. In section 6.6, the model of agent knowledge is introduced, while sections 6.8 and 6.9 describe the proposed agent protocol. Finally, section 6.10 presents the evaluation of our model and then the summary is presented in section 6.11.

6.4 Basic Definitions

This section presents a number of basic definitions which are closely related to web service composition, namely, Web service, User composition request, Web service composition and Service relations, as in the following:

**Definition 6.1.** Web Service is a functional unit, which transforms a set of input to a set of output parameters and can be represented by a tuple $WS = (op, I, O; QoS)$; where $op$ is the service operation name, $I$ is the set of input parameters of the service, $O$ is the set of output parameters of the service. $QoS$ is the set of quality parameters of the service, e.g., availability, execution time, price, and reputation. $I$ and $O$ are called the functional properties, while $QoS$ values are categorized as non-functional.

Usually, $op$, $I$, and $O$ are represented by semantical concepts (e.g., service ontology). The ontology is a common language on which a particular domain
community has agreed to represent a set of standard concepts within the domain such as bank industry.

**Definition 6.2.** *User Composition Request* is a request to compose a service and can be represented by a tuple \( R_u = (I_p, O_e; P_g, QoS_g) \), where \( I_p \) is the provided input set, \( O_e \) is the expected output set, \( P_g \) is the given personalized preferences and \( QoS_g \) is the given Quality of Service constraints.

\( P_g \) is a set of personalized preferences that reflects user’s interests. For instance, in a typical transportation scenario, a user prefers to go by plane over a car if the distance is more than 600 kilometers.

**Definition 6.3.** *Web Service Composition:* given that a service community \( S_c = \{ws_1, ws_2, ..., ws_n\} \), where \( ws_i = (op_i, I_i, O_i; QoS_i) \), and a user composition request \( R_u = (I_p, O_e; P_g, QoS_g) \), the web service composition is defined as the process of finding a finite set of services \( S_{R_u} \) from \( S_c \), i.e., \( S_{R_u} \subseteq S_c \) where:

1. Services in \( S_{R_u} \) can be organized in a certain order to provide a new value-added service \( ws_x \);
2. The \( ws_x \) accepts \( I_p \) and produces \( O_e \) as input and output sets, respectively;
3. The \( ws_x \) is supposed to meet user’s specified set of preferences;
4. The \( ws_x \) and its sub-services should meet user’s specified QoS constraints \( QoS_g \).

**Definition 6.4.** *Service Relation* is a relation that describes the order of execution among a set of services. Given two services, \( service_1 \) and \( service_2 \), the basic relations can be specified as follows:
• **Sequence**: service\(_2\) is executed after the completion of service\(_1\). It can be denoted as the sequence operator (\(\triangleright\)).

• **Mutual exclusion**: Either service\(_1\) or service\(_2\) is executed, but not both. It can be denoted as the mutual exclusion operator (\(\otimes\)).

• **Parallelism**: Both service\(_1\) and service\(_2\) are executed concurrently. It can be denoted as the parallelism operator (\(\oplus\)).

• **Iteration**: service\(_1\) is executed several times. It can be denoted as the iteration operator (\(*\)).

The composition approach generates these basic relations during the composition process. For example, if the user requests alternative services, the algorithm then searches for different options and hard codes them using (\(\otimes\)) symbol within the final composition workflow. Figure 6.1 demonstrates basic sample scenarios for the above relations.

### 6.5 WSC Scenarios

As user’s needs are always changing in today’s Cyber World, utilizing available services to compose new ones with different user preferences and contexts becomes a promising solution. In this section, multiple sample scenarios are presented where Web service composition provides an effective way to respond to user’s requests:

• **Online Shopping Scenario** is widely used as an example of a web service composition problem. In reality, many online shopping websites apply their shopping model as a series of core tasks that decompose the main task, i.e., buying a product online. Typically such a scenario includes product
searching, order submitting, cost payment and arrangement of shipping. By dedicating a service for each task, they can be integrated into a workflow template, i.e., represented a composite WS.

Although the scenario seems simple, the actual process model can be refined in multiple ways. For instance, a user might have a preference for the payment method based on his current account balance. If there is enough money, he wants to pay in full, otherwise, in instalments. This means that two different process models are available to fulfil the main task. Traditional web service composition systems often ignore addressing such a problem due
Chapter 6: Agent-Based Model To WSC

to the fact that their main focus is on the planning process itself. In addi-
tion, inquiring user balance can not be performed unless a real invocation is
made at runtime, which means the final process model will be selected later.
Therefore, the WSC system must merge both models into one workflow
template and leave the actual selection to runtime service selection module.

Figure 6.2 presents the process model of this scenario. Using service relation
symbols, the final workflow of the composite service can be: (((S1 ⊳ S2) ⊕ 
(S3)) ⊳ (S4 ⊗ S5) ⊳ (S6)). It has a parallelism operation between S2 and S3
as a necessity to include both, while a mutual exclusion operation between
S4 and S5 as an alternative selection.

In fact, dealing with user preferences within planning is an additional stage.
Thus, reaching higher user satisfaction by generating more personalized com-
posite services is an important feature. This can be achieved by incorporat-
ing the process of handling user preferences into the planning process.

• A trip planning scenario, including hotel and transportation arrange-
ments, in which the user selection of transportation means depends on the
actual distance between the source and destination cities. In other words,
he might prefer to take an airplane when the distance is longer than 500
km or rent a car when it is less. The difference between this example and
the previous is that the process model of the later can be determined at
composition time, not at runtime, if the system has the ability to acquire
the distance through a real invocation of a distance service. In agent based
composition systems, the scenario can be achieved by delegating one of the
agents to invoke such services at composition time. This shows the advan-
tage of adopting an agent based planning approach which provides a high
level of dynamicity to the problem of personalized web service composition.
Another common scenario is **book obtaining task**. It can be realized by two main tasks, through borrowing from the library or buying online. As the user is eager to read the book, he first prefers to borrow it from the library. If it is not available or the service fails for any reason, then he will
buy it online with the fastest shipping service. From this scenario, it is clear that the user has a certain priority on the different services and he expects the system to take it into account when suggesting him a composite service. Therefore, the user will provide the system with his preferences along with an initial request.

6.6 Modelling of Agent Knowledge

In this section, the knowledge model proposed in this thesis is presented. First, AND/OR graph model is introduced as the underlying model to represent service and agents dependencies. Then, the decomposition process of service dependency graph is explained. Finally, the internal knowledge base within each agent is described.

6.6.1 AND/OR graph

In WSC, the dependencies among web services are usually modeled into a graph structure such as the AND/OR graph. It is a special type of graph which is commonly adopted in artificial intelligence, as a data structure used in automatic problem solving. An AND/OR graph is considered a generalization of a directed graph where a number of nodes are connected by a set of generalized edges or connectors, with the following characteristics:

1. Each connector connects a set of nodes $v_i$, $i = 1..n$ to a single node $v_j$.

2. Connectors have two types: AND connectors and OR connectors depending on the type of logical relationship among $v_i$. 

143
3. Based on that, each node can be classified as one of three different types: AND node, OR node, or AND/OR node. AND node and OR node are the nodes that have only an AND connector or OR connector, respectively; while AND/OR node is the one which has a logical OR relationship among multiple AND connectors (i.e., different AND alternatives)\[88]\.

### 6.6.2 AND/OR graph and Web Services

The WSC problem dictates that a composite WS is actually a group of other atomic/composite web services which can be invoked in a certain order. Various web services can provide same functionality from different service providers; also more than one WS might be required to provide the input parameters to the composite service. Such dependency relations can be effectively mapped to corresponding connectors in an AND/OR graph. Hence, exploiting the well-identified AND/OR graph within WSC model effectively facilitates the composition process. Due to the existence of a strong analogy between AND/OR graph and WSC problem, we adopt AND/OR graph theory to design our model. In this model, each agent represents a particular node with which other nodes (or agents) can be connected by AND, OR, or both connectors. Practically, AND/OR graph is transformed to a simplified version in which each node receive only a single type of connector, i.e., AND or OR connector. This means that all AND/OR nodes are simply converted to an OR node connected to a set of dummy nodes each of which receives one original AND connector (i.e., AND alternative) \[88]\.

However, as agents are self-managed entities, it is not necessary to perform that transformation. Instead, each agent will store and manage all connectors it has regardless of their type. This can be achieved by maintaining a structured list of connectors within the agent’s knowledge base. Once the agent wants to assess
Chapter 6: Agent-Based Model To WSC

the status of a connector, the knowledge base will be inquired, and an appropriate assessment is then performed. In particular, if it is an OR connector, the agent will check if any of the connected agents is ready to provide the information, while in case of AND connector, the agent must ensure that all the connected agents are set to deliver their information. This mechanism adds another advantage to the adoption of Agent technology with AND/OR graphs in order to facilitate the composition of new web services.

6.6.3 Service Dependency AND/OR Graph

Service Dependency Graph (SDG) is a directed graph that captures dependencies among services within a particular service community. Typically, the dependency relations are constructed based on the functional characteristics of services, i.e., input and output parameters. Following the paradigm of producer-consumer, service \( s_a \) is called dependent on service \( s_b \) if service \( s_b \) produces one or more output parameters that service \( s_a \) consumes. This work proposes the use of AND/OR graph to represent dependency relations among services, which can be called Service Dependency AND/OR Graph. In this graph, services are labelled as AND nodes, while parameters, whether of input or output type, are modelled as OR nodes. Its basic formulation can be summarized as follows:

- \( N = \{x | x \text{ is a node in } G\} \),
  where \( N \) is the set of all nodes in the AND/OR graph \( G \).

- \( E = \{(x, y) | x \in N \land y \in N \land \text{there is a connector from } x \text{ to } y\} \),
  where \( E \) is the set of all edges over \( G \).

- \( P = \{x | x \text{ is a service parameter}\} \),
  where \( P \) is the set of all service parameters.
• \( S = \{ x \mid x \text{ is a service} \} \),
  where \( S \) is the set of all services.

• \( I = \{ x \mid (x \in P) \land (x \text{ is given by user}) \} \),
  where \( I \) is the set of input parameters given by the user.

• \( C(n) = \{ x \mid \exists (n, x) \in E \} \),
  where \( C(n) \) is the set of all child nodes of node \( n \).

Furthermore, the Graph has the following properties:

1. \( \forall x \in N, (\varphi(x) = \text{’OR’}) \lor (\varphi(x) = \text{’AND’}) \),
   This indicates that all nodes are of two types, AND and OR nodes. There
   is no node that has both types of connectors, i.e., AND/OR node.

2. AND nodes represent services, while OR nodes represent parameters.

\[
\varphi(n) = \begin{cases} 
  \text{OR} & \text{if } n \in P \\
  \text{AND} & \text{if } n \in S 
\end{cases}
\]
   where \( \varphi(n) \) is a function that determines the type of a node.

3. \( \forall (x, y) \in E, \varphi(x) \neq \varphi(y) \),
   This demonstrates that no direct edge between two nodes of similar type.
4. Each node $n$ is labelled such that:

$$\Gamma(n) = \begin{cases} 
\text{satisfied} & \forall x \in C(n), \varphi(x) = 'OR' \\
& \land \Gamma(x) = 'known' \\
\text{unsatisfied} & \exists x \in C(n), \varphi(x) = 'OR' \\
& \land \Gamma(x) = 'unknown' \\
\text{known} & (\exists x \in C(n), \varphi(x) = 'AND') \\
& \land \Gamma(x) = 'satisfied') \lor (n \notin I) \\
\text{unknown} & (\forall x \in C(n), \varphi(x) = 'AND') \\
& \land \Gamma(x) = 'unsatisfied') \land (n \notin I)
\end{cases}$$

where $\Gamma(x)$ is the function that determines the label type of node $x$. Each AND node is considered satisfied if and only if all of its child OR nodes are labeled as known. In contrary, it is unsatisfied if and only if at least one of its child nodes is unknown. On the other hand, each OR node is labeled as known if and only if at least one of its child AND nodes is satisfied, or it is already provided by the user. While it is considered unknown if and only if all of its child nodes are not yet satisfied and also it is not given by the user.

5. Each edge $(x, y)$ is labelled such that:

$$\Upsilon(x, y) = \begin{cases} 
\text{active} & \text{if } x \text{ is likely obtainable from } y \\
\text{inactive} & \text{if } x \text{ is not obtainable from } y
\end{cases}$$

where $\Upsilon(x)$ is the function that determines the label type of the edge from parameter OR node $x$ to service AND node $y$. Such an edge is considered
active if it is not yet examined as a part of the solution. In contrary, it is 
inactive if service $x$ is already queried and it failed to participate in the final 
composition.

Figure 6.3 demonstrates a sample SDG following the above formulations. Notably, 
each Service AND node is illustrated by a Rectangular shape while Service Param-
eter OR nodes are circled. Being a directed graph, the directional edges usually 
connect a Service AND node to a number of Service Parameter nodes where each 
of which represents one of its input parameters. Additionally, there are some leaf 
nodes of both types AND or OR, where the former is a service that does not need 
any input and the later is an ordinary input parameter which does not have a 
service to provide.

By using SDG AND/OR Model, the service composition process can be viewed 
as the task of finding a sub-graph within the overall SDG graph. That is, using 
user provided inputs, the top services of such a graph can generate the desired 
output, given that both sets of input and output parameters are specified in the 
initial user query.

In classical centralized WSC approaches, the composition process is applied 
directly to the entire SDG. However, as each service in the proposed model is 
assigned to a software agent, thereby, the SDG is decomposed into smaller sub-
graphs which are distributed among the agent community members to be stored 
within their knowledge bases.
6.6.4 SDG decomposition

Due to the adoption of Agent-based composition approach, the goal of this process is to decompose the original Service Dependency Graph into a number of subgraphs among the agent community. Initially, each service AND node is assigned to an agent which is then called a Service Agent (SA). Consequently, the assigned service node, its child parameter OR nodes with their direct providing service nodes, are all transferred to SA’s internal knowledge base. Figure 6.4 depicts how the decomposition process is applied where each service AND node is assigned to a software agent, shown surrounded with a green boundary. Based on that, the agent now is responsible to two main tasks; the first is to handle all the inquiries about this particular service while ensuring its applicability in regard to
Chapter 6: Agent-Based Model To WSC

the provided information. Also, it carries on an internal reasoning process to assist in the overall composition. In the following sections, these tasks are explained in details.

For the sake of simplicity, each agent is assigned a single service, i.e., singular-delegation. However, they can certainly handle more services which is then called a multiple-delegation. As a result of the decomposition process, WSC task becomes a planning task of finding a sub-graph within the overall decomposed SDG. This needs reliable collaboration among the agent community members through a pre-defined protocol. In following sections, the protocol and its algorithms are presented in more details.

Figure 6.4: SDG Decomposition
6.6.5 Agent knowledge base Model

Agent knowledge base model is identified as a subgraph of SDG that captures the functional profile of the web service alongside its direct dependencies on other services. As illustrated in Figure 6.5, the model consists of three levels with the following characteristics:

- The 1st level has the Root node which is a service AND node that the agent represents and is responsible to manage.

- The 2nd level has a number of parameter OR nodes equals to the number of input parameters of the service, where each OR node represents one parameter.

- The 3rd level has a number of terminal service AND nodes each of which represents a service that can provide the parameter of its parent OR node.

- Edges from 2nd level to 3rd level can be either active or inactive.

- OR nodes have two statuses, i.e., known and unknown.

- AND nodes have two statuses, i.e., satisfied and unsatisfied.

An active edge from an OR node to a terminal AND node indicates that the OR parameter is likely obtainable by corresponding AND service. That is service agents has not been queried yet. In contrast, if the edge is inactive, it means that service agent has failed to participate in providing the OR parameter and hence no need to ask it for assistance.

As explained in section 6.6.3, a known status for an OR node indicates that this parameter is already available and hence no need to ask any agent for assistance. The parameter can be available in two cases: (1) the parameter is originally
received from the end user as part of the input parameter list within his query. (2) the parameter is already provided through some agent as a result of the recent communication session. On the other hand, unknown status can be assigned to an OR parameter node when the parameter has been given yet neither by end user nor by other agents. Section 6.9 presents how this model is utilized during web service composition in more details.

6.7 Agent Life Cycle

The service agent can have one of three states at any given time during the composition process, namely, unvisited, visited, and reasoning. Being in a status of unvisited means that the Agent has not been searched yet, which can happen in two cases. The first is when the system performs the start-up initialization, while the second is during the reinitialization process upon receiving a new composition request from the user.

The state of visited, on the other hand, is assigned to the Agent when the search for the current request has completed either successfully or not. It means that the
Agent has reached a final decision and therefore it can immediately confirm or disconfirm its capability to solve any further requests.

The last state, *reasoning*, is declared by the Agent when the current request cannot be fully satisfied based only on Agent’s internal capability, while it knows that other depending agent(s) might assist in finding a solution. Therefore, it starts to consults these agents which their capabilities are captured in its own knowledge. During this phase, the agent does not respond to any request but instead memorizes them until it reaches a final decision. Figure 6.6 shows a state diagram for service agent.

### 6.8 Agent Messages

During the composition, different types of messages are communicated among the agent community in order to facilitate the planning for the requested composite
service. As we use JADEX [90] as a platform on which our framework is prototyped, a subset of the standard Agent Communication Language (ACL) [91] is adopted, i.e., Call For Proposal (CFP), PROPOSE, REQUEST, CONFIRM and DISCONFIRM, illustrated in Table 6.1. The following is a brief description of each message.

- **CFP**: the purpose of this message is to ask for a proposal to solve a new task. Upon receiving a composition request from the end user, User Agent, $A_{usr}$, usually sends this message to all participating service agents through utilizing a central agent directory service. Through receiving such a message by a service agent, two actions can be accomplished, notifying the receiver agent about the arrival of a new composition task, and allowing the agent to reset its internal status and knowledge settings for a possible coming request. Practically, depending on the configuration of MAS, the request can be issued directly by the User Agent or indirectly from another mediator agent, during the composition process. The role of the mediator agent is to help reduce the number of CFP messages needed. In such a configuration, this agent can pre-determine a subset of potential cooperative agents for a particular task, perhaps by analysing agent interaction logs, and thus a list of potential

<table>
<thead>
<tr>
<th>Message</th>
<th>Sender</th>
<th>Receiver</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFP</td>
<td>$A_{usr}$</td>
<td>$A_{all}$</td>
<td>action = $\text{Compose}(R^i, R^o)$</td>
</tr>
<tr>
<td>PROPOSE</td>
<td>$A_i$</td>
<td>$A_{usr}$</td>
<td>action = $\text{canProvide}(X_s, ws_i)$</td>
</tr>
<tr>
<td>REQUEST</td>
<td>$A_{usr}$</td>
<td>$A_i$</td>
<td>action = $\text{provide}(X_s, ws_i, ws_c)$</td>
</tr>
<tr>
<td>REQUEST</td>
<td>$A_i$</td>
<td>$A_i$</td>
<td>action = $\text{provide}(X_s, ws_i, ws_j)$</td>
</tr>
<tr>
<td>CONFIRM</td>
<td>$A_i$</td>
<td>$A_{usr}$</td>
<td>workFlow,$X_s, ws_i, ws_c$</td>
</tr>
<tr>
<td>DISCONFIRM</td>
<td>$A_i$</td>
<td>$A_{usr}$</td>
<td>$X_s, ws_i, ws_c$</td>
</tr>
<tr>
<td>CONFIRM</td>
<td>$A_i$</td>
<td>$A_j$</td>
<td>workFlow,$X_s, ws_i, ws_j$</td>
</tr>
<tr>
<td>DISCONFIRM</td>
<td>$A_i$</td>
<td>$A_j$</td>
<td>$X_s, ws_i, ws_j$</td>
</tr>
</tbody>
</table>

Table 6.1: Messages
receivers is formed to which the current CFP message is to be sent. This approach effectively reduces the number of CFP messages required for each composition request. Since this type of message has the largest number of requests (as it will be shown in complexity analysis section 6.10.1).

- **PROPOSE**: PROPOSE message acts as a response to a previously received CFP message by proposing to perform a given task. Specifically, the service agent informs User Agent, the sender of CFP, that it can partially or fully provide the required parameter list $X_s$ through the output parameter(s) of service $ws_i$ it represents. Specifically, this message informs the user agent if the service agent is able to provide all parameters that the user agent has asked for or only some of them. However, if none, then the service agent prefers not to respond indicating it is unable to assist in fulfilling this request. This passive approach is crucial to reduce the communication messages by discarding unnecessary incapability-declaring messages. However, the shortcoming of such approach is that not clearly declaring incapability makes the user agent assumes its certainty, while it may happen due to other reasons such as connectivity issues, time-out thresholds, agent business with other requests. Overall, reducing communication costs is preferred over handling such special cases, as the agent working environment is always very dynamic.

- **REQUEST**: Using this message, the sender requests the receiver to perform a specific action. The content part describes the action to be performed, i.e., provide the list of parameters $X_s$ by verifying the capability of service $ws_i$. The REQUEST message has two types, the *first* is used when the $A_{usr}$ has already selected an $A_i$ as a candidate agent to provide a required parameter list. This happens upon receiving a PROPOSE message from $A_i$, expressing the willingness to assist. While the *second* type is self-generated by $A_i$ during the internal reasoning phase. This occurs when the agent cannot fulfill the
request by itself and has enough knowledge about whom he can ask for assistance.

- **CONFIRM**: This message enables the sender to inform the receiver that the requested action can be successfully performed. It also can be divided into two subtypes, depending on receiver type. If the receiver is $A_{usr}$, it affirms that $A_i$ acknowledges the ability to provide $X_s$ through the output parameter(s) of service $ws_i$. A complete workflow plan, including $ws_i$, is also provided. The second type is communicated when the receiver is a peer agent within the community. Hence, $A_j$ will be informed that the input parameters of $ws_j$ can be provided by the matching outputs $X_s$ of $ws_i$ along with a complete workflow plan, including $ws_i$.

- **DISCONFIRM**: By this message, the sender informs the receiver that the requested action cannot be performed. That is, there was no successful workflow which its output list can provide the inquired parameter list $X_s$. Similar to CONFIRM, DISCONFIRM falls into two subtypes, depending on receiver type, i.e., $A_{usr}$ and a peer agent. In both cases, it indicates that the reasoning phase has failed to find a solution. This suggests that the receiver should start to seek for another peer agent for assistance.

### 6.9 Agent-Based WSC Algorithm

The algorithm starts upon receiving a request from the user for composing a new web service. As a result, a User Agent is created to represent the user and take the responsibility of finding a composition solution if any exists. This agent cooperates with other service agents and ask them for assistance. As a starting point of that
cooperation, the User Agent issues a CFP message and then sends it to the whole agent community.

■ **Upon receiving CFP**: The first step, by which a service agent is engaged in a composition process, is when it receives a CFP message, typically from a User Agent. The agent starts by resetting its internal status to *unvisited*, in addition to other necessary initialization steps such as resetting its logs. Secondly, it assigns the status of *known* to all OR nodes, which are initially provided with a user request (Algorithm 6.1, lines 3-7). This step is important to avoid unnecessary communications to find out a service that can provide such nodes while they are already available by the user. In other words, if a certain input parameter is given, there is no need to seek for an agent to generate it.

Then, the main goal output of the agent $Goal_{A_i}.output$, where $Goal_{A_i}$ is represented by input/output pair of parameter sets of the web service it represents, is compared with the user query output set $R^o$ (Algorithm 6.1, line 8). The availability of a subset of query output within the goal output indicates that this agent can positively participate in the attempt of solving the requested task. Consequently, it sends a PROPOSE message to User Agent informing it about its intended participation (Algorithm 6.1, lines 9-11). Otherwise, the agent stays passive in response to the original CFP message and waits for a possible involvement in the intermediate phase of the composition task. This step is intended to reduce the communication messages by ignoring to reply if the service agent cannot participate, which then lowers the overall complexity of the algorithm.
Algorithm 6.1. When Agent $A_i$ receives a CFP message - $\text{CFP}(A_{\text{usr}}, A_i, \text{action} = \text{COMPOSE}(R^i, R^i))$

1: $A_i.\text{status} \leftarrow \text{unvisited}$;
2: reset all messages logs and counters;
3: for each $OR_i$ node do
4: if $OR_i \in R^i$ then
5: $OR_i.\text{status} = \text{known}$
6: end if
7: end for
8: $X_s \leftarrow \text{Goal}_{A_i}.\text{output} \cap R^c$;
9: if $X_s$ not Empty then
10: send $\text{PROPOSE}(A_i, A_{\text{usr}}, \text{action} = \text{CanProvide}(X_s, w_{si}))$;
11: end if

Upon receiving PROPOSE: When the User Agent receives all PROPOSE messages from the service agent community, algorithm 6.2 is executed. Theoretically, the number of PROPOSE messages equals the number of CFP messages sent in the prior step, however, it is typically less due to the passive strategy adopted when no participation is possible. $A_{\text{usr}}$ starts by constructing its internal knowledge ($\text{BelG}_{A_{\text{usr}}}$). This step is needed since each query is likely to have different details, i.e., input, output and possibly preferences. As a result, various agents might offer their capabilities which $A_{\text{usr}}$ needs to remember and subsequently to reason. As aforementioned, this knowledge

Algorithm 6.2. When $A_{\text{usr}}$ receives All PROPOSE messages

1: $\text{BelG}_{A_{\text{usr}}} \leftarrow \text{ConstructKnowledge}()$;
2: if $\text{BelG}_{A_{\text{usr}}}$ not Empty then
3: $S_p \leftarrow \text{GetSolution}(\text{BelG}_{A_{\text{usr}}})$
4: for each $(A_k; w_{sk}; X_{sk}) \in S_p$ do
5: $\text{Req} \leftarrow \text{REQUEST}(A_{\text{usr}}, A_k, \text{action} = \text{Provide}(X_{sk}, w_{sk}, w_{sc}))$;
6: send $\text{Req}$;
7: $\text{SRLog}_{A_{\text{usr}}} \leftarrow \text{SRLog}_{A_{\text{usr}}} \cup \text{Req}$;
8: end for
9: else
10: Inform the user that request is unachievable;
11: end if
will be modelled in the form of AND/OR graph; where the user agent then checks whether there is a possible solution or not by reasoning $BelG_{A_{usr}}$.

The reasoning process is achieved by invoking the $GetSolution$ function (Algorithm 6.2, line 3), presented in the next paragraph. Once a solution is found, which is a list of services that can collectively satisfy all the output in the original request, $A_{usr}$ sends a REQUEST message to each in-solution service. $A_{usr}$ then remembers all requests thereby it can afterwards match them with their corresponding replies (Algorithm 6.2, lines 4-8). If a solution cannot be found, $A_{usr}$ reports to the user that his request to compose the Web service is unachievable. This case is when at least one of the output parameters required in the original user request cannot be generated by all available service agents. This situation might indicate that the user needs to review and update his request by perhaps providing more input information to facilitate the next composition task. It might also notify the system administrator of possible investigation of dependency relations among service agents and/or suggest adding new individual services to help find a solution for this case.
Algorithm 6.3. Generate a candidate solution (GetSolution)

Input:
BelG, Agent belief represented as an AND/OR graph

Output:
Solution, a set of candidate web services with parameters they can provide.

Function:
1: for each $OR_i$ node in BelG do
2:   if $OR_i$.status = unknown then
3:     $c \leftarrow$ BestTerminalNode($OR_i$);
4:     if $c$ not null then
5:       for all parents($OR_j$) of $c$ do
6:         $OR_j$.status $\leftarrow$ known;
7:         $S = S \cup \{OR_j\}$;
8:       end for
9:   else
10:      return null;
11:   end if
12:   $Solution = Solution \cup \{c.A_i; c.ws_i; S\};$
13: end if
14: return $Solution$;

GetSolution is the function responsible for performing the reasoning process on the Agent’s knowledge base. Its primary purpose is to try to extract a potential solution from Agent’s knowledge base for a given request. That means forming a list of web services that is sufficient (alongside the initially provided input parameters by the user request) to provide a complete set of input parameters.

In fact, the feasibility of this solution is not determined until these services are checked during the reasoning process. More precisely, the function checks the status of all parameter OR nodes (Algorithm 6.3, line 2), i.e., all input parameters of current service, and attempt to set their status to known. This is achieved by selecting one of its child terminal nodes, i.e., services, based on certain criteria applied by the sub-function BestTerminalNode() (Algorithm
6.3, line 3), discussed in a following paragraph. Once a particular terminal node is chosen and added to the potential solution, all of its own parent OR nodes are labeled as known (Algorithm 6.3, lines 5-8). Which has a twofold advantage, first is to ensure that none of the parameters is double provided by more than one service in the solution to be formed and second is to minimize the number of messages to be communicated with the services of the solution. This technique shows how AND/OR graph basic operations can effectively facilitate the composition process.

At the end, the function will return a solution as a set of subsets, each of which consists of a service id and a parameter list it can generate (Algorithm 6.3, line 12). Each subset is to be used to create a request message during the composition process. Otherwise, the function fails and returns a null value indicating that there is no potential solution to this task. Efficiently, this function immediately stops whenever it cannot satisfy any OR node, meaning that the underlying parameters has no candidate service to generate. This function is also called during the internal reasoning process performed within the algorithms of REQUEST, CONFIRM and DISCONFIRM messages as it will be shown in the next sections.

**BestTerminalNode** (BTN) is a function responsible for selecting the best terminal node, given a specific OR parent node. Concretely, it chooses the web service with the highest rank among all services that can produce a particular output parameter. First, it computes the cost of all child services for the given parameter (initially, the number of in-degree is used as a cost measure), and then it chooses the service with the maximum magnitude, discriminating the inactive individuals. However, the mechanism of calculating the cost measure of each web service can easily be updated within this function.
Given an \( OR_i \) node and a set of its child nodes \( C_{w^s} \), the score of \( OR_i \), its children, and best terminal node (BTN) can be computed such that:

\[
C_{w^s} = \{c_{w^s}^1, c_{w^s}^2, \ldots, c_{w^s}^k\}, \quad k = |\text{deg}^+(OR_i)|
\]

\[
\text{score}(c_{w^s}^j) = \begin{cases} 
0 & \text{if } n \text{ is a user agent node (start node)} \\
\text{deg}^-(c_{w^s}^j) & \text{if } n \text{ is a service agent node}
\end{cases}
\]

\[
\text{score}(OR_i) = \max \{\text{score}(c_{w^s}^j) \times w_{OR_i}^j\}
\]

\[
\text{BTN}(OR_i) = \begin{cases} 
c_{w^s}^j & \text{if } \text{score}(c_{w^s}^j) = \text{score}(OR_i) \\
\text{null} & \text{otherwise}
\end{cases} \wedge w_{OR_i}^j = 1
\]

where \( c_{w^s}^j \) is a terminal AND node within the knowledge base of the agent. The \( \text{deg}^-(c_{w^s}^j) \) and \( \text{deg}^+(c_{w^s}^j) \) are the in-degree and out-degree of node \( c_{w^s}^j \) respectively, while \( w_{OR_i}^j \) is the weight magnitude of the edge connecting the parent \( OR_i \) node to its child terminal AND node \( c_{w^s}^j \). From the definition, it is noticeable that BTN function selects the active terminal node with the highest rank based on the number of incoming links.

In Figure 6.7, the selection process of the best terminal node for our running example is depicted. For simplicity, the set of possible solution sets to only service agent \( S_0 \) is shown. The process begins when GetSolution function is
Figure 6.7: A Sample Belief Knowledge Base of a Service Agent, Showing Selection Process of BestTerminalNode Function.

invoked, which in turn inter-calls the BestTerminalNode function. According to the in-degree cost measure, the list of rankings for all of $S_0$’s child nodes is presented in Table 6.2, where $S_1$ has the highest rank of 3 as it can provide all 3 parameters (i.e., $p_1$, $p_2$ and $p_3$) to $S_0$. In addition, the ranks of $S_2$ and $S_3$ are both a magnitude of 2, since both can produce only 2 of the parameters needed, i.e., $p_1$ and $p_2$ by $S_2$ and $p_2$ and $p_3$ by $S_3$, respectively.

Table 6.2: Numeric Terms of Sample Services

<table>
<thead>
<tr>
<th></th>
<th>$score(c_{ws})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>3</td>
</tr>
<tr>
<td>$S_2$</td>
<td>2</td>
</tr>
<tr>
<td>$S_3$</td>
<td>2</td>
</tr>
</tbody>
</table>

Intuitively, the BTN function favours the node with the highest number of output parameters according to the internal knowledge of current service
agent. This policy serves the main goal of the whole composition algorithm of minimizing the communication cost among the agent community involved in a composition task, where the agent chooses the minimal service set of possible solution sets that are available to produce the complete list of its input parameters. To further explain the idea, the complete set of possible solution sets to provide $p_1$, $p_2$ and $p_3$, the output parameters of $S_1$, $S_2$ and $S_3$ collectively and at the same time the input parameters of $S_0$, is listed as follows:

$$\text{Possible Solution Sets} = \{S_1\}, \quad \text{len} = 1$$
$$\{S_1, S_2\}, \quad \text{len} = 2$$
$$\{S_1, S_3\}, \quad \text{len} = 2$$
$$\{S_1, S_2, S_3\}, \quad \text{len} = 3$$
$$\{S_2, S_3\}, \quad \text{len} = 2$$

Obviously, all repeated solution sets are removed from the overall possible solution set, resulting into 5 possible solution sets. In addition, the best solution set is the first set which has only one service element ($\{S_1\}$) that can provide all the parameters. Consequently, $\{S_0\}$ will communicate only with $\{S_1\}$ leading to the minimal communication cost available.

However, assuming that $\{S_1\}$ is unable to run due to the lack of availability and/or support from depending agents and since the policy used enforces the exclusion of any failing service(s) from the previous solution, the complete solution set that is generated by GetSolution function will be:

$$\text{Actual Solution Set} = \{S_1\}$$
$$\{S_2, S_3\}$$

The above example demonstrates the effectiveness of the internal reasoning model within the agent knowledge. In particular, the immediate exclusiveness of unsuccessful services along with their future opportunities to assist
in providing more parameters has two main advantages. First, it achieves an effective reduction of the number of solution sets leading to a faster approach to reaching a final decision for a potential solution. This discriminates all redundant solutions. Furthermore, it also ensures generating minimal possible distinct solutions, as shown in the previous example. Therefore, that reasoning model supports the goal of minimizing solution sets number and solution sets sizes leading to the main goal of the model which is minimizing communication costs among the agent community. By satisfying that, the model is expected to generate applicable workflows at a reduced communication cost.

■ Upon receiving REQUEST: When service agent $A_i$ receives a request message, Algorithm 6.4 is executed. This algorithm essentially maintains two types of information, i.e., internal memory and status, while it sends three different types of messages, i.e., REQUEST, CONFIRM and DISCONFIRM according to its internal decision.

For its memory, the service agent records all REQUEST messages which it has received into $RRLog_{A_i}$, the number of REQUEST messages it has sent into $SR_{cnt_{A_i}}$, and the set of service solution for current service requirement into $AS_p$, respectively. For status, $A_i$ declares the status of unvisited meaning that this is the first time that a REQUEST message is received, and the search for this service is about to start. The second status the agent can show is visited, which indicates that the agent just finished the search for this service. However, a third status, reasoning, is set by the agent if the request cannot be instantly fulfilled; while based on the former two statuses, a final decision and reply can be reached. When the status of $A_i$ is unvisited, three cases are distinguished:

- Case 1: The input parameters of the request $R^i$ includes all the input parameters of Agent’s goal $Goal_{A_i}$ (Algorithm 6.4, lines 3-7). Since the
Algorithm 6.4. Upon receiving a REQUEST messages, \(\text{REQUEST}(\text{Sender}, A_i, \text{action} = \text{Provide}(X_s, ws_i, ws_j))\)

1: \(\text{RRLog}_{A_i} \leftarrow \text{RRLog}_{A_i} \cup \text{REQUEST}(\text{Sender}, A_i, \text{action} = \text{Provide}(X_s, ws_i, ws_j))\);

2: if \(A_i\text{'status} = \text{unvisited}\) then
3: if \(\text{Goal}_{A_i}\text{'input} \subseteq R^i\) then
4: \(A_i\text{'status} \leftarrow \text{visited};\)
5: \(A_i\text{'feasible} \leftarrow \text{true};\)
6: \(A_i\text{'workFlow} \leftarrow ws_i;\)
7: send \(\text{CONFIRM}(A_i, \text{Sender}, (A_i\text{'workFlow}, X_s, ws_i, ws_j))\);
8: else
9: \(AS_p \leftarrow \text{GetSolution}(BelG_{A_i})\)
10: if \(AS_p = \text{null}\) then
11: \(A_i\text{'status} = \text{visited};\)
12: \(A_i\text{'feasible} \leftarrow \text{false};\)
13: send \(\text{DISCONFIRM}(A_i, \text{Sender}, (X_s, ws_i, ws_j))\);
14: else
15: \(A_i\text{'status} \leftarrow \text{Reasoning};\)
16: for each \((A_k; ws_k; X_{sk}) \in AS_p\) do
17: \(\text{Req} \leftarrow \text{REQUEST}(A_i, A_k, \text{action} = \text{Provide}(X_{sk}, ws_k, ws_i));\)
18: send \(\text{Req};\)
19: \(SR^m_{Ai} \leftarrow SR^m_{Ai} + 1;\)
20: end for
21: end if
22: end if
23: else
24: if \(A_i\text{'status} = \text{visited}\) then
25: if \(A_i\text{'feasible} = \text{true}\) then
26: send \(\text{CONFIRM}(A_i, \text{Sender}, (A_i\text{'workFlow}, X_s, ws_i, ws_j));\)
27: else
28: send \(\text{DISCONFIRM}(A_i, \text{Sender}, (X_s, ws_i, ws_j));\)
29: end if
30: end if
31: end if
input that the agent needs is fully satisfied by the request input, the agent’s status is set to \textit{visited} with positive \textit{feasibility} and the search is successfully finished by sending CONFIRM message \((CONFIRM(A_i, \text{Sender}, (A_i.workFlow, X_s, ws_i, ws_j)))\) to the \textit{Sender} of the REQUEST message.

- **Case 2:** The input parameters of the request \(R^i\) do not fully include all the input parameters of Agent’s goal \(Goal_{A_i}\) (Algorithm 6.4, lines 10-13), and a set of alternative service solution cannot be formed. This implies that \(A_i\) cannot satisfy the request neither based on its own capability nor on its supporting agents’. Hence, the agent’s status is set to \textit{visited} with negative \textit{feasibility} and the search is unsuccessfully finished by sending DISCONFIRM message \((DISCONFIRM(A_i, \text{Sender}, (X_s, ws_i, ws_j)))\) to the \textit{Sender} of the REQUEST message.

- **Case 3:** The input parameters of the request \(R^i\) does not fully include all the input parameters of Agent’s goal \(Goal_{A_i}\) (Algorithm 6.4, lines 15-20), and the set of alternative service solution can be formed. This indicates that the agent might be able to satisfy the request based on the capabilities of its supporting agents. Hence, the agent’s status is set to \textit{reasoning} while postponing the decision on \textit{feasibility} to a subsequent time. The search on the set of alternative service solution is initiated by sending REQUEST messages \((REQUEST(A_i, A_k, action = Provide(X_s, ws_i, ws_j)))\) to each service participating in the solution.

In contrast, if the Agent has the status of \textit{visited} (Algorithm 6.4, lines 24-30), i.e., it receives a REQUEST \((Sender, A_i, action = Provide(X_s, ws_i, ws_j))\) message after a prior search process has completed with a final decision, then either a CONFIRM message \((CONFIRM(A_i, \text{Sender}, (A_i.workFlow, X_s, ws_i, ws_j)))\)
\( X_s, ws_i, ws_j \)) or a DISCONFIRM message (\( \text{DISCONFIRM}(A_i, Sender, X_s, ws_i, ws_j) \)) is sent to \( Sender \) based on the feasibility status of \( A_i \).

- **Upon receiving DISCONFIRM**: When \( A_i \) receives a DISCONFIRM message, as a reply to a previously sent REQUEST message, it is told that the web service associated with its REQUEST cannot participate in the solution. Its role now is to seek for another solution that considers the incapability of the failing service along with its effects on all related parameters. More specifically, it must ensure that this service is excluded from all further searches for a new solution. This can be achieved by deactivating all the edges of the failing terminal node with its OR parent nodes, i.e., this service and its output parameters respectively (Algorithm 6.5, lines 7-9). In addition, all its parent nodes included in \( X_s \) must also be reset to unknown and thereby offered another chance to be provided through an entirely different service (Algorithm 6.5, lines 10-12).

However, before generating a new service solution, the agent ensures that all replies, whether positive or negative, are received (Algorithm 6.5, line 5). This is important to maintain a proper utilization of agent knowledge such that an entirely new decision is made after consulting all services in the current solution. Otherwise, there might be a wrong overlap between a subsequent pair of generated solutions. The agent checks whether the number of sent REQUEST messages \( SR^\text{cnt}_{A_i} \) is equal to the sum of the lengths of \( RaLog_{A_i} \) and \( RaLog_{A_i}^{\text{DisC}} \) (Algorithm 6.5, line 8). If all received, a new candidate solution \( AS_p \) is generated by calling \( \text{GetSolution}(BelG_{A_i}) \) function. If a solution exists, the counter of sent REQUEST messages is updated, i.e., decreased by the number of DISCONFIRM messages and \( |RaLog_{A_i}^{\text{DisC}}| \) is reset.
Chapter 6: Agent-Based Model To WSC

The search on the set of alternative service solution is then initiated by sending REQUEST messages (REQUEST($A_i, A_k$, action = Provide($X_{sk}, w_{sk}, w_{si}$))) to each service participating in the solution (Algorithm 6.5, lines 18-22). If no solution is found, then the agent is ready to announce its final status as visited with negative feasibility and inform all requesters about its decision. That is, it sends a CONFIRM message (CONFIRM($A_i$, Sender, ($A_i$.workFlow $\triangleright$ $w_{si}, X_{sk}, w_{si}, w_{sk}$))).
Algorithm 6.5. Upon receiving a DISCONFIRM message, DISCONFIRM($A_j, A_i,(X_s, w_{sj}, w_{si})$)

1: $T_{msg} \leftarrow \lvert RaLog_{Ai} \rvert + \lvert RaLog_{Ai}^{DisC} \rvert$;
2: if $T_{msg} \neq SR_{cnt}^{A_i}$ then
3: $RaLog_{Ai}^{DisC} \leftarrow RaLog_{Ai}^{DisC} \cup$ DISCONFIRM($A_j, A_i,(X_s, w_{sj}, w_{si})$);
4: end if
5: if $T_{msg} \geq (SR_{cnt}^{A_i} - 1)$ then
6: for each DISCONFIRM($A_{jk}, A_i,(X_{sk}, w_{sjk}, w_{si})$) ∈ $RaLog_{Ai}^{DisC}$ do
7: for all parents($OR_\alpha$) of $w_{sjk}$ do
8: $E_{w_{sjk}} \leftarrow$ Inactive;
9: end for
10: for each $OR_\beta$ in $X_{sk}$ do
11: $OR_\beta$.status ← unknown;
12: end for
13: end for
14: $AS_p \leftarrow$ GetSolution($BelG_{A_i}$)
15: if $AS_p$ not null then
16: $SR_{cnt}^{A_i} \leftarrow SR_{cnt}^{A_i} - \lvert RaLog_{Ai}^{DisC} \rvert$;
17: $\lvert RaLog_{Ai}^{DisC} \rvert = \{\}$;
18: for each ($A_k; w_{sk}; X_{sk}$) ∈ $AS_p$ do
19: $Req \leftarrow$ REQUEST($A_i, A_k, action = $Provide($X_{sk}, w_{sk}, w_{si})$);
20: send $Req$;
21: $SR_{cnt}^{A_i} \leftarrow SR_{cnt}^{A_i} + 1$;
22: end for
23: else
24: $A_i$.status ← visited;
25: $A_i$.feasible ← false;
26: if $A_i$.workFlow not Empty then
27: for each REQUEST($Sender, A_i, action = $Provide($X_{sk}, w_{si}, w_{sjk})$) ∈ $RRLog_{A_i}$ do
28: send CONFIRM($A_i, Sender, (A_i, workFlow \triangleright w_{si}, X_{sk}, w_{si}, w_{sjk})$);
29: end for
30: else
31: for each REQUEST($Sender, A_i, action = $Provide($X_{sk}, w_{si}, w_{sjk})$) ∈ $RRLog_{A_i}$ do
32: send DISCONFIRM($A_i, Sender, (X_{sk}, w_{si}, w_{sjk})$);
33: end for
34: end if
35: end if
36: end if
if a solution workflow is already found (i.e., this round failed to find an alternative solution in mutual exclusion case); otherwise, it sends DISCONFIRM message \(\text{DISCONFIRM}\left(A_i, \text{Sender}, (X_{sk}, ws_i, ws_{jk})\right)\) to all of them (Algorithm 6.5, lines 24-34).

Upon receiving CONFIRM: When Service Agent \(A_i\) receives a CONFIRM message, Algorithm 6.6 is executed. Similar to REQUEST message algorithm, this algorithm mainly maintains two type of information, i.e., internal memory and status, while it sends only two different types of messages, i.e., REQUEST and CONFIRM according to its internal decision.

For its memory, the service agent records all CONFIRM messages which it has received into \(RaLog_{A_i}\), the number of CONFIRM messages it has sent into \(SR_{cunt}^{A_i}\), and the set of a new service solution for current service requirement into \(AS_p\), respectively. For status, the agent announces the status of visited by overwriting the previous reasoning status, meaning that this is the end of the search for this service with a successful solution. The second status the agent can maintain is reasoning which indicates that the agent just started looking into another solution. Unlike the reasoning status of REQUEST, which is set by the agent if the request cannot be immediately satisfied, it is now maintained for finding another solution besides the current successful one. That is to satisfy the initial request of multiple solutions, expressed by mutual exclusion operation.
Algorithm 6.6. Upon receiving a CONFIRM messages, CONFIRM($A_j,A_i,(workFlow,X_s,ws_j,ws_i)$)

1: $RaLog_{A_i} \leftarrow RaLog_{A_i} \cup CONFIRM(A_i,A_j,(workFlow,X_s,ws_i,ws_j))$;
2: if $|RaLog_{A_i}| + |RaLog_{DisC}^{A_i}| = SR_{cnt}^{A_i}$ then
3:   if $A_i.$workFlow not Empty then
4:      $A_i.$workFlow $\otimes \leftarrow \{\}$;
5:   end if
6:   if $|RaLog_{A_i}| = 0$ then
7:      if $A_i.$workFlow not Empty then
8:         $A_i.$workFlow $\leftarrow \{RaLog_{A_i}[1].workFlow\}$;
9:   else
10:      for each $CONFIRM(A_{jk},A_i,(workFlow_p,X_{sk},ws_{jk},ws_i)) \in RaLog_{A_i}$ do
11:         $tWorkFlow \oplus \leftarrow \{workFlow_p\}$;
12:      end for
13:      $A_i.$workFlow $\leftarrow \{tWorkFlow\}$;
14:   end if
15:   AS_p $\leftarrow$ null;
16:   if Multiple solution is still required then
17:      reset all messages logs and counters;
18:      for each $CONFIRM(A_{jk},A_i,(workFlow_p,X_{sk},ws_{jk},ws_i)) \in RaLog_{A_i}$ do
19:         for all parents($OR_{\alpha})$ of $ws_{jk}$ do
20:            $E_{ws_{jk}} \leftarrow$ Inactive;
21:         end for
22:         for each $OR_{\beta}$ in $X_{sk}$ do
23:            $OR_{\beta}.$status $\leftarrow$ unknown;
24:         end for
25:      end for
26:      AS_p $\leftarrow$ GetSolution($BelG_{A_i}$)
27:      if AS_p not null then
28:         for each $(A_k; ws_k; X_k) \in AS_p$ do
29:            $Req \leftarrow REQUEST(A_i,A_k,\text{action} = Provide(X_{sk}, ws_k, ws_i))$;
30:            send Req;
31:            $SR_{cnt}^{A_i} \leftarrow SR_{cnt}^{A_i} + 1$;
32:         end for
33:      end if
34:   end if
35:   if AS_p is null then
36:      $A_i.$status $\leftarrow$ visited;
37:      $A_i.$feasible $\leftarrow$ true;
38:      for each $REQUEST(Sender,A_i,\text{action} = Provide(X_{sk}, ws_i, ws_{jk})) \in RRLog_{A_i}$ do
39:         send CONFIRM($A_i,Sender,(A_i,\text{workFlow} \triangleright ws_i, X_{sk}, ws_i, ws_{jk}$));
40:      end for
41:   end if
42: else
43:      call DISCONFIRM procedure;
44:   end if
45: end if
Chapter 6: Agent-Based Model To WSC

The agent waits for all CONFIRM messages of current service solution to be received before taking a final decision and inform its requesters. This is achieved by comparing the size of $RaLog_{A_i}$ to the number of REQUEST messages sent previously $SR_{cnt}^{A_i}$ (Algorithm 6.6, line 2-3). Since receiving a CONFIRM message demonstrates that its sender can provide the requested parameters to $A_i$, having all CONFIRM messages received acknowledges the positive feasibility of current service solution. Therefore, it is the time to send a final confirmation message in response to all REQUEST messages received by the Agent. Recall that if any of request messages cannot be confirmed, i.e., in response to which a DISCONFIRM message is received, the DISCONFIRM algorithm handles that and restarts another search for a new solution.

The final plan workflow of the CONFIRM message can be created and categorized into three cases. In the first, the workflow consists of only one dependent service (Algorithm 6.6, line 7). While the second is when more than one service participates in the solution, hence a parallelism operation is presented to join these services at the same level (Algorithm 6.6, lines 10-13). Finally, the new workflow is created by inserting a sequence operation between the workflow and the current agent service (Algorithm 6.6, line 39). Once the workflow is created, the CONFIRM message is ready to be sent to all requesters (Algorithm 6.6, lines 38-40).

The third case occurs when multiple solutions are required (Algorithm 6.6, lines 16-34), which is expressed by a mutual exclusion service operation. Therefore, a new search for another solution needs to be carried out, before sending the final CONFIRM message. The new search is started by resetting message logs and counters. $A_i$ then excludes the current solution by deactivating all edges from which it was formed, plus, to resetting the status of all
OR nodes in $X_s$ in order to be included in the new solution (Algorithm 6.6, lines 18-25). Although the agent now has a solution and is able to announce its status as *visited*, the status remains *reasoning* to signal that it is still searching for another solution.

Once the resetting process is completed, a new candidate solution $AS_p$ is requested by calling $GetSolution(BelG_{A_i})$. If a solution exists, the search on the set of alternative service solution is initiated by sending REQUEST messages ($REQUEST(A_i, A_k, action = Provide(X_{sk}, w_{sk}, w_{si}))$) to each service participating in the new solution. Once the search is finished, whether after generating a multiple or single solution, the agent announces its final status and informs all requesters accordingly. More precisely, the agent’s status is set to *visited* with positive *feasibility* and the search is finished successfully by sending CONFIRM messages ($CONFIRM(A_i, Sender, (A_i,workFlow\triangleright w_{si}, X_{sk}, w_{si}, w_{sjk}))$) to all requesters.

Both CONFIRM and DISCONFIRM algorithms \(^1\) for User Agent are similar to their corresponding ones of service agent except the final step. That is, instead of addressing the requester agent, $A_{usr}$ informs the end user of the success or failure of the composition process.

### 6.9.1 Illustrative Planning Scenario

This section demonstrates how the Agent planning protocol is utilized to compose a web service following the Online Shopping Scenario, as illustrated in Figure 6.8.

When processing this composition request, different cases can exist based on several factors such as the availability of agents, user input parameters, in addition

\(^1\) Appendix A. presents both algorithms.
to user preferences. Certainly, agent availability is crucial since each service is
delegated to an individual agent. It is assumed that all agents are ready and all
initial input parameters are also provided by the user. To simplify the idea, it is
also assumed that the user prefers paying in cash if he has enough cash in his bank
account; if not, then he would choose to pay in instalments instead.

Figure 6.9 shows the communications among the Agent community members
upon receiving user’s request. The red rectangle surrounds communications re-
quired among agents for the first case, while the blue rectangle is highlighting the
second. Both cases end when $A_6$ sends back a CONFIRM message to the user
agent. However, if the user is willing to receive a multi-option workflow solution,
then both sub-workflows will be generated, integrated into one workflow and fi-
nally sent to him. In fact, this is considered the third case. Consequently, the
user may receive one of the following workflows:

\[ \diamond \text{Case 1: } (((S_1 \triangleright S_2) \oplus (S_4)) \triangleright (S_5)) \triangleright (S_6)). \]
Case 2: \(((S_1 \triangleright S_2) \oplus (S_3)) \triangleright (S_4) \triangleright (S_6))\)

Case 3: \(((S_1 \triangleright S_2) \oplus (S_3)) \triangleright (S_4 \otimes S_5) \triangleright (S_6))\)

This scenario exhibits some interesting features of the proposed planning Agent algorithm in terms of the numbers of messages communicated along with the reduction strategy of the overall communication cost.

Notably, the number of messages required for case 1 is eight while it is six messages for case 2. These are the messages needed in addition to user agent’s messages which are six CFP messages to the whole community and a triple of PROPOSE, REQUEST, and CONFIRM/DISCONFIRM communicated with \(A_6\), the shipping service agent.

In addition, it is noticed that the final agent (i.e., \(A_1\)) is only inquired once during the composition of case 1 but not case 2. This is due to the fact that agent \(A_2\) is already inquired during case 1 which in turn reached its final decision (i.e., has visited status with certain feasibility) by consulting \(A_1\). Therefore, \(A_2\) does not need any reasoning and is able to respond immediately to any REQUEST message. This strategy helps ensure not to re-consult a previously consulted agent during the same composition process and thereby effectively reduce the communication cost.

\section*{6.10 Model Evaluation}

In this section, the evaluation of the proposed Agent-based composition model is presented. First, the time complexity of the model is analysed which is then followed by a set of experiments to validate the model.
Figure 6.9: Agent Communications of Online Shopping Scenario.


6.10.1 Complexity Analysis

As the agent-based approach to web service composition is used, which follows the distribution paradigm of problem solving, we adopt the number of messages communicated during the composition process as an evaluation metric to the time complexity of our algorithm. Given a service dependency graph $SDG$ consisting of $n$ service agents and $r$ dependency relations (although $r$ can theoretically reach $n^2$ if there is a relation between every pair of agents, it is practically unrealistic) and composition request $Cr$, the number of messages required to find a composite service can be estimated as in the following:

1. Since $CFP$ messages are sent to all service agents, once a service request is received, the total number of $CFP$ messages is $n$. However, only a few service agents respond by proposing to participate in performing the task. These agents partially or fully match their output parameters with the output parameters of the service request. Thus, the number of $CFP$ messages is far more than that of the $PROPOSE$ messages which indicates that the number of messages required in the initialization step is about $n$.

2. In the worst case scenario, the whole $SDG$ is considered, i.e., all possible subgraph solutions will be explored, which means that all relations of $SDG$ are searched. As a result, the number of $REQUEST$ messages sent is $r$, while the number of corresponding replies, i.e., $CONFIRM$ and $DISCONFIRM$ messages, is also $r$.

3. In total, the number of messages needed to achieve a composition task is estimated to $n + 2r$ in the worst case.

One way to cope with the large number of $CFP$ messages is that the system might be able to pre-determine a subset of potential cooperative agents for a particular
task, perhaps by analysing agent interaction logs. Thus a list of potential receivers is formed to which the current CFP message is to be sent. This approach effectively reduces the number of CFP messages required for each composition request. In addition, as mentioned above, the number of responding agents with a PROPOSE message is practically far less than the number of agents received CFP messages; therefore, the search space is already reduced under the $2r$ worst case limit. This case is only imaginable if all services are serving one final service which is an unrealistic case.

6.10.2 Experiments AND Discussion

A number of experiments is conducted to evaluate the performance of the proposed composition model. All experiments are implemented in JADEX platform and performed on a PC computer with a 64-bit Windows 7 system, 8 GB RAM, and Intel® Core™ i7 CPU 720. The performance is measured in terms of communication cost, i.e., the number of messages required to generate a final workflow. To assess the effectiveness of the algorithm, a comparison with a common search strategy that is widely used in comparative studies of graph search algorithms, i.e., Brute Force search (BF) is conducted, where all possible solutions are searched. In all experiments, the number of alternative solutions is set to 1.

6.10.2.1 Dataset

To evaluate the proposed WSC algorithm, a publicly available dataset ICEBE05 [92] is used, which was essentially used as a benchmark test data for the web service challenge at ICEBE 2005. The dataset consists of two sets of test data, namely $Composition_1$ and $Composition_2$, respectively. They differ in the degree of complexity such that the latter is more complicated, while they share a similar design.
Both of them include nine test sets of repositories of web service specifications, containing different numbers of WSDL files. The WSDL files are stored in an abstract form, showing mainly input and output parameters within request and response messages tags respectively. In addition, each test set is equipped with 11 queries in form of provided input and required output messages. The following experiments, however, are conducted on Composition\textsubscript{1} group.

In order to verify the validity of composition algorithms, all test sets are varied according to three aspects, (1) Size of web service community, (2) Number of input and output parameters, and (3) Complexity of composition solution. More specifically, the size of web service community is assigned three levels, i.e., 2156, 2656, and 4156. Similarly, the number of input and output parameters of services is set to three levels, expressed in ranges of 4-8, 16-20, and 32-36. While the complexity of composition solution, with regards to the available queries, is varied into four levels, i.e., 1, 25, 125, and 625.

Figure 6.10 shows a sample Belief knowledge base of an agent, where the service that it represents is realized as a root AND node, i.e., service\textsubscript{p00a0084929}, illustrated in a clear rectangular shape. The service has four input parameters, namely, \textsubscript{p03a0622587}, \textsubscript{p63a1766560}, \textsubscript{p19a0291388}, \textsubscript{p20a5634703}, which are shown as OR data nodes. Finally, each parameter node has a list of services that generates its value, represented as a set of terminal nodes each of which is a child node to its parent OR node. It is noticeable that more than one OR node may share a certain terminal node, resulting in creating a cycle relation within the graph.
Figure 6.10: A Sample Belief Knowledge Base of a Service Agent
6.10.2.2 Experiments

Figure 6.11 shows a comparison between our approach and Brute Force approach. The experiment conducted on Composition test set (1-20-4), where the x-axis represents the query number, while the y-axis refers to the number of messages communicated during each composition. Results show that our approach outperforms BF with a large margin in the number of messages. Although Brute Force approach can find the optimal solution out of all (i.e., in case the quality of the workflow is desired), it is expensive and non-scalable. In contrast, our approach is more affordable and scalable in terms of the communication cost. Furthermore, as each query has a certain solution complexity, initially designed by dataset generator, the results indicate a positive relationship between this complexity and the number of messages. In particular, query 1,2,4,9 are the highest in communication cost, while queries 3,5,7 have the least, with the exception of query 11 which has only one service in the final workflow, i.e., no composition is required.

Figure 6.12 demonstrates the result of the second experiment, in which the communication cost of the proposed approach versus the service population size is studied. The experiment is conducted on Composition test sets (1-20-16, 1-50-16, 1-100-16), where the x-axis represents the query number, while the y-axis refers to the number of messages communicated during each composition. It is noticeable that whenever the service population size increases, the communication cost also increases. This is reasonable for the following reasons. (1) Any increase in the number of service agents will need more CFP messages to send out upon receiving a new user query. (2) Having more agents raises the probability of sharing similar input and output parameters among a wider group of agents in the community, which means that certain parameter OR nodes will have extra edges within the
SDG. This definitely increases the number of alternative sub-solutions considered within agent inter-reasoning process, which also raises the communication cost.
Chapter 6: Agent-Based Model To WSC

Figure 6.12: Communication Cost for Different Agent Community Sizes

<table>
<thead>
<tr>
<th>Query #</th>
<th>1-20-16</th>
<th>1-50-16</th>
<th>1-100-16</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2169</td>
<td>2669</td>
<td>4169</td>
</tr>
<tr>
<td>2</td>
<td>2169</td>
<td>2669</td>
<td>4169</td>
</tr>
<tr>
<td>3</td>
<td>2165</td>
<td>2665</td>
<td>4165</td>
</tr>
<tr>
<td>4</td>
<td>2169</td>
<td>2669</td>
<td>4169</td>
</tr>
<tr>
<td>5</td>
<td>2165</td>
<td>2665</td>
<td>4165</td>
</tr>
<tr>
<td>6</td>
<td>2159</td>
<td>2659</td>
<td>4159</td>
</tr>
<tr>
<td>7</td>
<td>2165</td>
<td>2665</td>
<td>4165</td>
</tr>
<tr>
<td>8</td>
<td>2167</td>
<td>2667</td>
<td>4167</td>
</tr>
<tr>
<td>9</td>
<td>2169</td>
<td>2669</td>
<td>4169</td>
</tr>
<tr>
<td>10</td>
<td>2165</td>
<td>2665</td>
<td>4165</td>
</tr>
<tr>
<td>11</td>
<td>2167</td>
<td>2667</td>
<td>4167</td>
</tr>
</tbody>
</table>
6.11 Conclusion

Fulfilling user’s needs through recommending web services is increasingly receiving attention from both industrial and academic parties. As such, when no individual service is available, recommending a composite service becomes the solution. Consequently, the emphasis of this chapter was to design a dynamic and cost effective composition model for web services. Due to the shortcomings of current centralized WSC approaches of suffering from performance bottleneck and single point of failure, this work adopts an agent planning based approach to service composition. In particular, this chapter answered the following questions mentioned at the beginning of the chapter.

The first question is, How to recommend composite web services in a complexity-balanced manner? This work exploits the analogy of Agent technology and Service computing in terms of working environment, i.e., the decentralized environment. On the other hand, to effectively manage and infer relationships among services, the well-established AND/OR Graph theory is utilized to construct Service Dependency Graph. To adapt this graph to Agent approach, it is distributed among the members of the Agent community, such that each agent becomes a delegate to an assigned service. Furthermore, the agent is responsible to manage and reason its own knowledge base that contains a portion of the original SDG. Upon receiving a user composition request, agents perform internal reasoning and corporate attempting to find a solution. To accomplish that, an ACL-like communication protocol including communication messages with their algorithms is presented. A key feature of this protocol is the reasoning strategy performed by the agent internally where it prioritizes the alternative workflow with the lowest communication cost and also ensures immediate exclusion of any failing agents. Such a strategy
generates the best solution in terms of communication cost within the agent internal search space and thereby the final workflow will be of quality close to optimal. In fact, to ensure finding the best workflow, a global reasoning approach must be used where an algorithm with a backtracking step is needed, e.g., A* algorithm. However, we abandoned this approach due to its high communication cost which is infeasible in Agent based approaches.

The second question answered in this chapter is, *How effective is the proposed composition approach?* To measure the effectiveness of the proposed decentralized composition approach the complexity analysis is first discussed. Then a set of experiments is conducted on a public composition dataset with different complexity levels. Results show the effectiveness of the proposed approach in terms of reducing the communication cost while guaranteeing finding a solution if one exists.
Chapter 7

Summary and Future Works

This thesis has proposed a recommendation framework for Web services that is capable of recommending individual and composite services. For individual service recommendation, an Integrated-Model QoS-based Graph is proposed that integrates both user-service ratings with users and service similarities using Jaccard coefficient. Top-k Random Walk algorithm is then applied to generate final recommendations. For composite service recommendation, an AI planning Agent-based service Model is proposed where Agents collaboratively cooperate through internal reasoning and external communication to attempt finding a cost effective workflow solution to current composition request. The evaluation showed that both models perform will in terms of recommendation accuracy and composition complexity respectively.

This chapter concludes this thesis by summarizing the research performed, highlighting the key contributions and listing future research in the related area.
Chapter 7: Summary And Future Works

7.1 Thesis Summary

This research concentrated on proposing a recommendation model not only for individual Web Services but also for composite Web services, towards the establishment of a comprehensive recommendation framework for web services. Because the popularity of Web services, especially with the rising of Cloud Based systems, is rapidly growing for implementing a unified interfaces to system’s functionalities, more attention is required from both academic and industrial sections to tackle serious implementation obstacles such as scalability, accuracy and dealing with lack of user feedback information in order to recommend proper and robust web services well-aligned with user preferences.

In chapter 2, a summary of the background information of recommender system models and approaches is presented. After overviewing the most popular approaches, including content-based, collaborative filtering and hybrid methods, the strengths and limitations of collaborative filtering methods (being the most widely adopted) are identified. Based on that, an insight to address some of those limitations is then introduced. In particular, the main issue which this research is focused on is the data sparsity issue which forms, with other issues, the whole Web service recommendation problem. As the proposed recommendation model is based on Graph theory, the recent recommendation techniques, including dimensionality reduction and Graph based methods, are then reviewed.

From the literature review conducted in Chapter 3, it can be seen that enhancing current web service recommendation models whether of individually- or compositely-oriented categories is important for meeting the rising adoption of web services besides the ever changing user preferences and contexts with recent technological advancements such as smart cities, human-computer-interaction, and Internet of Things.
Chapter 7: Summary And Future Works

On individually-oriented service recommendation, this thesis took advantage of the graph based recommendation model with Random Walk algorithm for its reputation in dealing with an essential problem, i.e., the sparsity problem. However, being aware of the deficiency of this model in terms of recommendation accuracy, the thesis proposes a similarity-based upgrading to the model to effectively deal with the accuracy while benefiting of data sparsity treatment.

On compositely-oriented service recommendation, the thesis exploited the effectiveness of Agent technology which can achieve a high level of dynamicity and robustness that real-world Web service environment experiences. Being aware of the trade-offs that are necessary between the dynamicity, efficiency and the robustness of the usage of centralized and decentralized composition approaches, this research selected the Agent-based decentralized composition approach to ensure the dynamicity and robustness of the resulting composite services. This provides dynamicity and robustness for the searching needed in the composition process while guaranteeing finding a good solution if one exists along with reducing "human-in-the-loop" activities to the minimum, approaching to a full automation process.

Modelling the individual Web Service recommendation approach using Graph theory with Random Walk algorithm was presented in Chapter 4. The model formation included the following: (1) the Jaccard coefficient was proposed in several variants that are appropriate for WS recommendation. Being suitable for binary rating scales, web service ratings were transformed from n-ary to binary rating scale. (2) the chapter defined three types of the building-block graphs where the first is a User-Service Bipartite rating-based graph, the second is a Unipartite graph that capture similarities among Users and the third is another Unipartite graph that capture similarities among Services. (3) An Integrated-Model QoS-based Graph model for WS recommendation is constructed, in which
User-Service Bipartite Graph can be fused with User-Based and Service-Based Unipartite Graphs. (4) The model applies a Top-K Random Walk recommendation algorithm onto the Integrated-model where the algorithm, after performing a walk, filters out already user-rated services and then selects the best ones to be recommended to end user. The model was validated by conducting a set of extensive experiments on a real-world WS dataset, followed by a comprehensive analysis on the impact of various experimental parameters. Results showed that improved recommendation accuracy are obtained by utilizing the proposed model in different integration approaches while it becomes less prone to data sparsity issue.

The enhanced individual recommendation model is presented in Chapter 5 as follows. First, to minimize complexity of the proposed model, the Chapter introduced an enhanced recommendation model, in which RW algorithm is better guided using a selected subset of Jaccard similarities, instead of the entire similarity set, while maintaining a acceptable level of recommendation accuracy. Second, to assist in observing the behaviour of the model in response to different experimental settings, a new measure to study how much the proposed model is reduced compared to the original domain was then proposed. Third, the enhanced model is validated through conducting a group of experiments on a real-world WS dataset with a comprehensive analysis of the impact of various experimental parameters. Finally, to check whether the model can be used as a generic recommendation model in terms of application domain, a movie rating dataset along with a publicly available recommendation library (librec) are used to compare it with several common recommendation approaches using various evaluation metrics. Results demonstrated proved the applicability and effectiveness of the model as a generic recommendation model.

The service composition approach was presented in Chapter 6 as follows:
Chapter 7: Summary And Future Works

(1) the chapter introduces the distributed service dependency model based on AND/OR graph structure which effectively encodes the dependency relationships among services based on the analogy with the producer-consumer generic model. To fit the Agent-based approach, it is then decomposed and distributed among individual members of the Agent community. (2) Within the agent knowledge, the assigned portion of the dependency model is stored in a simplified AND/OR model where AND nodes are assigned to service agents while OR nodes to data parameters those agents consume or produce. (3) A reasoning approach was proposed which is designed to find quality sub-composition solutions in terms of applicability and communication cost. By exploiting well-established connections stored in Agent’s knowledge, the approach generates and favours the applicable composite solution with the least communication cost. (4) an Agent Communication Protocol for service composition, derived from ACL standard, is then presented. The protocol contains a short list of messages that agents utilise to coordinate their mutual efforts in fulfilling a composition task requested. Detailed algorithms of how the agent responds, either immediately or after consulting other agents, for all protocol messages are demonstrated and explained. (5) The chapter evaluates the proposed composition model using a public composition dataset and results indicated the effectiveness of the approach in terms of reducing communication cost needed to respond to given composition requests.

The main contributions of this research are: 1) an individual Web Service recommendation model using Graph structure with Top-k Random Walk algorithm is proposed. The model utilizes the Jaccard coefficient in several variants to create two Unipartite similarity-based graph that capture similarities among Users and among Services and then integrates them with the original user-service rating graph. The model achieves higher recommendation accuracy and stronger tolerance to data sparsity. 2) The model is investigated to potential reduction
for lowering time complexity by utilizing a thresholding technique. The model is further tested as a generic recommendation model through the use of an ordinary rating domain. 3) A service composition model based on Agent-technology is presented. In this model, a distributed service dependency model based on AND/OR graph is decomposed and distributed among service agents who are equipped with a well-defined internal reasoning mechanism based on agents’ knowledge. They use a communication protocol to collaborate to find a cost-effective workflow for a composition request. 4) The performance of the components of the proposed framework is verified using publicly available datasets, a recommendation library, and the JADEX multi-agent platform.
Chapter 7: Summary And Future Works

7.2 Future Work

1. For individual service recommendation model:

   (a) Conduct an investigating of other similarity measures within the proposed model in different application domains. As a result, present a study that highlights the findings and recommendations of which measures are beneficial in which application domains.

   (b) Expand the model to include more characteristics in addition to essential user-item ratings. For example, in a movie recommender model, a movie-actor participation information can be added as an additional dimension besides user-movie ratings. In service QoS-based model, this can be another QoS property, such as Throughput and Price in addition to Response Time, to be added to the model.

   (c) Take advantage of the flexibility of the model in terms of initial settings in order to simulate other recommendation approaches. This can be exploited in applying the Top-k Random Walk algorithm iteratively, for example, applying one round on the algorithm and then selecting the users with highest similarity magnitude and then lunch a second step where these users are selected initially in the input vector. This two-step algorithm simply simulate applying CF approach while taking advantage of Random Walk characteristics.

2. For composite service recommendation model:

   (a) Add the missing functionalities (some mentioned as assumptions in Chapter 1) for the model to be a complete composition system such as agent declaration of joining the system, the creation of the administration agent, and the handling of user requests.
(b) Propose to update the reasoning mechanism inside the agents in order to support higher level of intelligence. For example, currently the agent explores only its internal knowledge (local domain), while it can further look ahead beyond that trying to globalize its decision to a certain degree as far as the complexity of the system stays within an acceptable limit.

(c) Integrate the individual service recommendation model with the composition model such that the former can be implemented as a recommender agent responsible for providing suggestions during the composition process. This needs updating the communication algorithm accordingly.

(d) Handle user’s preferences, such user pre-selection of a specific service, during the composition. One way to accomplish that is by filtering out the final list of workflows at the end of the composition process. A more effective approach is to apply a task-decomposition strategy during the composition process where the first step is to locate the service agent of the user’s preference and then launch a two opposite-direction subtasks to explore the entire service dependency graph. This work also needs updating the communication protocol.

(e) Finally, consider the execution of the final workflow through exploiting the execution capability of agents. In this point, the distinguish between information-gathering and world-changing services must also be distinguish.

The above suggested future works are all directed to a realization of a comprehensive framework for web service recommendation which is the ultimate target of the work accomplished in this thesis.
Appendix A

User Agent

CONFIRM/DISCONFIRM Algorithms

Both CONFIRM Algorithm A.1 and DISCONFIRM Algorithm A.2 for User Agent are similar to their corresponding ones of service agent except the final step. That is, instead of addressing the requester agent, $A_{usr}$ informs the end user of the success or failure of the composition process.
Algorithm A.1. Upon receiving a CONFIRM messages, CONFIRM(A_j,A_usr,(workFlow,X_s,ws_j,ws_i))

1: RaLog\_A\_usr ← RaLog\_A\_usr ∪ CONFIRM(A\_usr,A\_j,(workFlow,X_s,ws\_j,ws\_i));
2: if |RaLog\_A\_usr| + |RaLog\_DisC\_A\_usr| = SR\text{cnt}\_A\_usr then
3:   if A\_usr.workFlow not Empty then
4:     A\_usr.workFlow ⊕ ← {};
5:   end if
6: if |RaLog\_A\_usr| = 0 then
7:   if A\_usr.workFlow ← {} then
8:     A\_usr.workFlow ← {RaLog\_A\_usr[1].workFlow};
9: else
10:   for each CONFIRM(A\_jk,A\_usr,(workFlow\_p,X\_sk,ws\_jk,ws\_i)) ∈ RaLog\_A\_usr do
11:     tWorkFlow ⊕ ← {workFlow\_p};
12:   end for
13:   A\_usr.workFlow ← {tWorkFlow};
14:   AS\_p ← null;
15: if Multiple solution is still required then
16:   reset all messages logs and counters;
17:   for each CONFIRM(A\_jk,A\_usr,(workFlow\_p,X\_sk,ws\_jk,ws\_i)) ∈ RaLog\_A\_usr do
18:     if all parents(OR\_α) of ws\_jk do
19:       E\_OR\_α \_ws\_jk ← Inactive;
20:     end for
21:   for each OR\_β in X\_sk do
22:     OR\_β.status ← unknown;
23:   end for
24:   AS\_p ← GetSolution(BelG\_A\_usr)
25: if AS\_p not null then
26:   for each (A\_k;ws\_k;X\_sk) ∈ AS\_p do
27:     Req ← REQUEST(A\_usr,A\_k,action = Provide(X\_sk,ws\_k,ws\_i));
28:     send Req;
29:     SR\text{cnt}\_A\_usr ← SR\text{cnt}\_A\_usr + 1;
30:   end for
31:   end if
32: if AS\_p is null then
33:     A\_usr.status ← visited;
34:     A\_usr.feasible ← true;
35:     for each REQUEST(Sender,A\_usr,action = Provide(X\_sk,ws\_i,ws\_jk)) ∈ RRLog\_A\_usr do
36:       Inform end user of the final workflow (A\_usr.workFlow ⊿ ws\_i);
37:     end for
38:     call DISCONFIRM procedure;
39:     call DISCONFIRM procedure;
40:     end if
41: else
42:     call DISCONFIRM procedure;
43:     end if
44: end if
Algorithm A.2. Upon receiving a DISCONFIRM message, DISCONFIRM($A_j,A_{usr},(X_s,ws_j,ws_i)$)

1: $Tmsg \leftarrow |RaLog_{A_{usr}}| + |RaLog^{DisC}_{A_{usr}}|$
2: if $Tmsg \neq SR_{cnt}^{A_{usr}}$ then
3: $RaLog^{DisC}_{A_{usr}} \leftarrow RaLog^{DisC}_{A_{usr}} \cup$ DISCONFIRM($A_j,A_{usr},(X_s,ws_j,ws_i)$);
4: end if
5: if $Tmsg \geq (SR_{cnt}^{A_{usr}} - 1)$ then
6: for each DISCONFIRM($A_{jk},A_{usr},(X_{sk},ws_{jk},ws_i)$) $\in RaLog^{DisC}_{A_{usr}}$ do
7: for all parents($OR_{\alpha}$) of $ws_{jk}$ do
8: $E^{OR_{\alpha}}_{ws_{jk}} \leftarrow$ Inactive;
9: end for
10: for each $OR_{\beta}$ in $X_{sk}$ do
11: $OR_{\beta}.status \leftarrow$ unknown;
12: end for
13: end for
14: $AS_{p} \leftarrow GetSolution(BelG_{A_{usr}})$
15: if $AS_{p}$ not null then
16: $SR_{cnt}^{A_{usr}} \leftarrow SR_{cnt}^{A_{usr}} - |RaLog^{DisC}_{A_{usr}}|$
17: $|RaLog^{DisC}_{A_{usr}}| = \{\}$;
18: for each ($A_k;ws_k;X_{sk}) \in AS_{p}$ do
19: $Req \leftarrow REQUEST(A_{usr},A_k,action = Provide(X_{sk},ws_k,ws_i))$;
20: send $Req$;
21: $SR_{cnt}^{A_{usr}} \leftarrow SR_{cnt}^{A_{usr}} + 1$;
22: end for
23: else
24: $A_{usr}.status \leftarrow$ visited;
25: $A_{usr}.feasible \leftarrow$ false;
26: if $A_{usr}.workFlow$ not Empty then
27: for each REQUEST($Sender.A_{usr},action = Provide(X_{sk},ws_i,ws_{jk})$) $\in RRLog_{A_{usr}}$ do
28: Inform end user of the final workflow ($A_{usr}.workFlow \triangleright ws_i$);
29: end for
30: else
31: for each REQUEST($Sender.A_{usr},action = Provide(X_{sk},ws_i,ws_{jk})$) $\in RRLog_{A_{usr}}$ do
32: Inform end user that his request cannot be fulfilled;
33: end for
34: end if
35: end if
36: end if
Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


