SEMANTIC-BASED MULTI-FEATURED RANKING ALGORITHM FOR SERVICES IN SERVICE-ORIENTED COMPUTING

Ammar Alsaig

A thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science

Concordia University

Montréal, Québec, Canada

November 2013

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mr. Ammar AbdulBasit Alsaig**

Entitled: **Semantic-Based, Multi-Featured Ranking Algorithm for Services in Service Oriented Computing**

and submitted in partial fulfillment of the requirements for the degree of

Master in Applied Science (Software Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________________ Chair
Dr. T. Popa

______________________________ Examiner
Dr. J. Paquet

______________________________ Examiner
Dr. O. Ormandjieva

______________________________ Co-supervisor
Dr. V. S. Alagar

______________________________ Co-supervisor
Dr. M. Mohammad

Approved by ______________________________
Chair of Department or Graduate Program Director

____
Dr. Christopher W. Trueman, Interim Dean
Faculty of Engineering and Computer Science

Date ______________________________

Abstract

Semantic-based Multi-featured Ranking Algorithm for Services in Service-oriented Computing

Ammar Alsaig

Service-Oriented Computing has brought great benefits for both service requesters and service providers. The potential of this paradigm cannot be achieved without efficient discovery and selection processes. The rapid-increasing volume of services and the heterogeneity of their features make the discovery and selection of services challenging. In this thesis we provide a novel vector-based ranking algorithm. The algorithm is both user-centric and semantic-based. It overcomes all restrictions and limitations that exist in previous vector-based ranking algorithms. We introduce fair ranking rules and apply them in our algorithm. The algorithm has been examined thoroughly with respect to its performance, accuracy and algorithmic complexity. We provide experimental results that show the significance and dominance of our solution over the existing ones.

Acknowledgments

Above all, I thank God for enlightening me with the knowledge and empowering me with the strength to complete this work. I thank him for surrounding me with kind people to only some of whom it is possible to give particular mention here. Without people around me, this thesis would not have been possible.

I am profoundly thankful and grateful for the support, teachings, and guidance of professor Vangalur Alagar. His knowledge and expertise in the field of mathematics and algorithms helped to build and improve the work in this thesis. Moreover, the guidance, support, and friendship of my co-supervisor , Dr. Mubarak Mohammad, has been valuable both academically and personally, for which I am extremely grateful.

I would like to extend my deepest thanks to my wife Duaa for her personal support and great patience at all times, it would have been a lonely road without her support. Without her and my children, Wedad , Sarah, and Yassir, I would not have been here. I also would like to express my thanks to my parents, and sisters for giving me their unequivocal support along the way, as always, for which my mere expression of thanks likewise does not suffice. A special thanks to my sister, Alaa who shared with me the difficulties and the joys of my years in Canada. She was a friend, a sister and a continuous support.

Last but not least, I would like to extend my gratitude to the Saudi Cultural Bureau for their financial support. I thank their staff for being always there for me to provide help.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Service-Oriented Computing (SOC) [SH06] has been well established as the main software development paradigm. It uses *Service* as the fundamental element for the development of applications in meeting social needs. SOC promotes development of distributed service applications in heterogeneous environments and provide customers to seek services with easy to use interfaces. An architectural model of SOC in which *service* is a first class element is called *Service-Oriented Architecture (SOA)* [Erl04]. The main elements of this architecture are *service registry*, *service planner*, *service provider* and *service requester*. A service provider prepares services and publishes them in service registry. A service provider can browse the contents of service registry, and then query the system for seeking services that match user queries. The planner is to deliver the services that match the user query. Given the large volume of services in the registry and the heterogeneous nature of services, finding the services that match a query according to a user's request and ranking them in a decreasing order of importance to the user are non-trivial tasks. There has to be an automatic and efficient method that can deliver ranked services to users based on their preferences. It is this problem that is addressed in this thesis.

A service can be described by many features. Typical features of a service are its functionality, its price, and other properties that describe the qualitative characteristics of the service. For example, an air travel service is described by its routing, price, and class of travel. Therefore, service discovery process is not based on one feature or criterion of the service, but rather on many features. These features, both qualitative and quantitative features, are heterogeneous in nature. A survey of services offered by current service-based systems in many domains, such as Health Care, Power Distribution, On-line Banking, and On-line Shopping, will convince us that people use them out of necessity but are dissatisfied with their performance. We use the term "performance" in a wider sense, to mean the ability of the system to provide users the facility to construct complex requests as well its ability to deliver services that best match the user requests. Both these user-centric essentials are almost non-existent in existing systems. We can say that the service models in these systems are awfully inadequate to express complex service requirements of clients, inept to adapt to changing contextual situations, and insufficient to meet the expected needs of clients. Above all, there is no accountability when the system does not meet user expectations. To remedy this situation Ibrahim [Ibr12] proposed a rich service model, and a framework for SOC. Ibrahim also studied many query types, methods for query processing, and proposed one method for ranking services. In this thesis, we focus on service ranking.

The vector-based ranking algorithm [Ibr12] is the first contribution in SOC for ranking services on multiple features. This algorithm is executed by the planning unit in the *FrSeC framework* [Ibr12]. In Chapter 3 we critically review and analyse all ranking algorithms that were known including Ibrahim's algorithm. After further scrutiny we found many limitations of all these algorithms. This motivated us to study and contribute to provide a solution that would remedy all the shortcomings and overcome the challenges faced by previous works in the literature of service ranking. The ranking algorithm developed in this thesis can be used in the planning unit of the FrSeC framework, as well as for ranking services in any service-oriented framework.

Our comprehensive study of the existing service ranking algorithms lead us to identify not only their inadequacies, but also lead us to the conviction that we need a ranking algorithm that is both user-centric and semantic-based. The user-centric semantic-based ranking algorithm that we have developed can be customized to work correctly for all application domains. The proposed ranking method will enable the requester to formulate a “request” that specifies the query, the preferred semantics, and preferences for seeking services that not only match the query features but also agrees with the specified semantics. The ranking is driven by the semantics specified in the user request. The algorithm will rank services in decreasing order of “satisfaction” to the specified semantics.

In general, no two consumers are likely to have the same set of preferences in selecting a service or buying a product. The set of preferences will depend on their personal opinions on the quality attributes of the service. Consequently, a service request must be made user-centric. As an example, a user might prefer cheapest air travel service between two destinations, whereas another user might prefer a non-stop service between the two destinations, regardless of ticket price. The user request contains a query that specifies the attributes of the service, and semantics. That is, a query will specify features that are to “match” features in selected services and a semantic will specify the ‘meaning’ (semantics) for matching. A user might require *exact match* of features whereas another user might prefer *best match* of features. Since the meaning of “match” itself may be different for different users we defined different modes, called “best match” and “exact match” in the system. These modes are in line with their common sense usage. We allow the user to submit such mode along with a query and semantic references within the request. In general, the semantic preferences that apply to each feature in the query will be part of ‘service request’. That is, the ranking method provides the users a wide range of different options to specify in order that they can tailor the results to meet their own needs. To the best of our knowledge, there is no method that considers all the mentioned perspectives.

1.0.1 Contributions

The significant contributions of this thesis are:

- A set of criteria for *fair ranking*.
- A literature survey and critical evaluation of existing ranking algorithms.
- A semantic-based ranking algorithm called the X-Algorithm.
- An extension of the X-Algorithm to rank composite services.

These contributions are presented in the following manner. In Chapter 2 we discuss fair ranking and present our fair ranking criteria. In Chapter 3 we evaluate the existing ranking algorithms with respect to fair ranking requirements. In Chapter 4 we present a conceptual user interface to be used with the X-Algorithm. Also, we present the phases of the X-Algorithm execution. In Chapter 5 we discuss the different semantics and options that are normally used by consumers in choosing and judging services in everyday life. Also, we present the X-Algorithm and outline its details with examples. In addition, we introduce compensation. In Chapter 6 we present an experimental results to examine accuracy, performance and complexity of our algorithm. In Chapter 7, we extend the X-Algorithm to rank composite services. In Chapter 8, we summarize our achievements and outline future extensions.

Chapter 2

Basic Concepts and Fair Ranking Requirements

In this chapter, we present the basic concepts of ranking algorithms. We first informally discuss how we perform implicit ranking of services in our daily life. Then, we explain how this implicit ranking should be made explicit when seeking online services. After that, we define basic terminologies involved with seeking online services. Also, we bring out some of the problems associated with ranking online services and we emphasize the need for a fair ranking algorithm, which does not exist today. Finally, we enumerate and briefly explain the requirements necessary to perform fair ranking.

2.1 Basic Concepts

In real life when we want to consume (buy) a service (product), we encounter many options. Then, we have to make a decision on which service (product) to choose. Services are described by the desired service features. For instance, when we book a flight some of the service features are number of transits to reach destination, in-flight food quality, and seat preference. Based on these features, users decide which airlines to choose.

In general, not all features are of the same concern to all consumers. Some consumers maybe interested in only a single feature, some others maybe interested in multiple features. This requires consumers to define features that capture their interest. For instance, when renting an apartment consumers have to define some points of interest such as price, quality, and location to help real estate agents to find them the best apartment that matches their requirements.

Requirements of consumers are diverse. Some consumers will have specific requirements that they are looking for, in that they want a service that matches exactly their requirement as much as possible. For instance, consumers may look for an exact number of "stars" when they want to book a hotel room. On the other hand, some consumers may be looking for cheaper price, additional amenities, and good dining quality. Some others may have a mix of both requirements, in the sense that some features are essential while some other offers are at an acceptable level. Another example is when consumers look for an apartment to rent, they usually have a specific requirement for the number of rooms, but they may compromise on location and rent amount.

On many instances some features may contradict others. This contradiction leads to a trade-off in final selection. It is accepted that a higher quality apartment implies a higher cost for the consumer. This trade-off is resolved by knowing which feature is more important for a consumer. A consumer may compromise the low price for a better quality if the quality feature is more important than the cost. In other words, the renter would be willing to trade the low cost features for features of higher quality. However, there is no objective decision that can be made automatically regarding level of importance, because every consumer may have different degree of importance attached to different features. Therefore, it is the responsibility of consumers to define the level of importance for each feature of the service they desire. For instance, it is the responsibility for the person who's looking for an apartment to inform the most important and the less important features to the real-estate agent. This will enable the agent to consider the most important features first, then following it the less important ones. Eventually, consumer definition of interesting features and features of importance would have an impact on ordering service selection.

We go through similar experience when we consume online services. We will encounter many options in selecting *Services*. In this thesis the emphasis on *Service* ranking and not on *Service* modeling. As such, we consider a *Service* to be a vector of features. Also, we usually have specific requirements which we formulate as a *Query*. Within a *Query* we attached a degree of importance, referred to as *Weight*, with each *Query* attribute. Finally, the available *Services* are *matched* with the *Query* to produce a list of *Services* that best fit our requirements in decreasing order of satisfaction. This process is referred to as *ranking*. Collectively, we refer to this process as *ranking of online services based on weighted query*. In order to rigorously explain this process algorithmically we define the concepts *Service*, *Query*, *Weight*, *Matching* and *Results* in the following sections.

2.1.1 Service

By *Services* we refer to the available options that we find in the market when we seek a *Service*. Each *Service* is described by single feature or multiple features. We refer to features of *Service* as attributes. Thus, a *Service* is defined as a list of attributes, where each attribute represents a feature of the *Service*.

$$Service = [attribute_1, attribute_2, \ldots, attributes_n]$$

Example 1. *Suppose a car rental service is described by the three features price, car_status, and deposit_price, we represent the car rental service as*

$$CarRentalService = [price, carStatus, depositPrice]$$

Because each *Service* is considered an alternative to the user, we use the terms *alternative* and *Service* interchangeably.

2.1.2 Query

Query is defined by consumers. It includes their specific requirements regarding a *Service*. Thus, a *Query* is represented as a list that includes one or more attributes. These attributes represent the favourable values of the features they want in the sought service. Hence, a *Query* is represented as

$$Query = [value_1, value_2, \dots, value_n]$$

Each value in *Query* list corresponds to one of the attributes in the *Service* definition. For instance, a specific *Query* for the $CarRentalService$ in Example 1 is

$$Query = [\$200, New, \$50]$$

where,

\$200 corresponds to $price$ feature of the service.

New corresponds to $carStatus$ feature.

\$50 corresponds to $depositPrice$ feature.

2.1.3 Weights

Weights refer to the level of importance associated with each value defined in the *Query*. As explained earlier, the importance of introducing weights is to resolve the trade-offs. The weight corresponding to a *Query* is a vector of values, such that there is a one-one correspondence between a favourable value of the *Query* vector and the value of the weight vector. Thus, the lengths of these vectors are equal. Each numerical value in the weight vector represents the level of importance that the consumer associates with the corresponding favourable feature value in *Query*. Hence, corresponding to a $Query = [value_1, value_2, \ldots, value_n]$ there exists a weights vector called

$$Weight = [W_1, W_2, \ldots, W_n]$$

The weight values in a weight vector can be given either numerically or as literals that can be interpreted by common sense semantics. As an example, the weight vector,

$$Weight = [LeastImportant, MostImportant, NotImportant]$$

may be used to specify the preferences of *Service* features. For numerical values, we use numbers in a specific scale, such as $[0-5]$, where 0 represents the lowest importance and 5 represents the highest importance. For the $CarRentalService$ shown in Example 1, the weights associated with the *Query* $Q = [\$200, New, \$50]$ can be $Weights = [5, 3, 0]$, where 5 is a weight associated with \$200 ($price$ feature), 3 is a weight associated with New ($carStatus$ feature), and 0 is a weight associated with \$50($depositPrice$ feature).

2.1.4 Matching

Matching is a core concept of the decision making process. This is because we make our decisions considering options that *satisfy* all or most of our requirements. However, this process is performed implicitly in real life. In contrast, in digital world, *Matching* should be accomplished algorithmically. Specifically, matching process puts a user *Query* against the set of all available *Services* and produces a *Matching Score*. This score is a number assigned to each feature of each *Service* and is interpreted as a means of rewarding a specific *Service* feature for matching the corresponding feature in *Query*. Matching Algorithms produce *Matching Scores* based on *Matching Rule/Matching Measure*, where a rule defines a specific reward or penalty when matching or mismatching occurs. Matching Scores related to one *Service* are eventually aggregated into one value. This value represents the *Matching* degree of each *Service* to *Query*. We determine the *Matching* degree by taking the weights of each feature. A simple method to calculate degree is to multiply each *Matching Score* by the corresponding weight. However, this simple method is insufficient to discriminate between *Services* that are relevant to a user *Query* .

Let us consider Example 1, in which each *Service* is described by the three features $price$, $carStatus$, and $depositPrice$. Assume we have three *Services*, S_1, S_2 and S_3 as three available *Services* in the market.

$$S_1 = [\$300, New, \$100]$$

$$S_2 = [\$200, UsedANDGoodCondition, \$60]$$

$$S_3 = [\$60, UsedANDBadCondition, \$10]$$

Consider $Q = [\ \$300,\ New,\ AnyValue]$ to be the consumer's *Query* and $W = [5,\ 3,\ 0]$ be the weights. In this case, the consumer is interested only in the features $price$ and $carStatus$. Let the *Matching Rule* be defined as follows:

$$MatchingRule = \begin{cases} MatchingScore = 1 & \text{if } (Q_i = S_i)(Matched) \\ MatchingScore = 0 & \text{if } (Q_i \neq S_i)(Mismatched) \end{cases}$$

By putting Q against the available *Services*, we can intuitively conclude that S_1 is the choice. Practically, this conclusion has been drawn based on *Matching* concept, where each element in the *Query* list is compared with the corresponding element in each *Service* list. Thus, by applying the defined Matching Rule on *Query* with each *Service*, we find out the following results

Query	*Weights*	*Service*	*Matching Scores*
$[\$300, New, AnyValue]$	$[5,3,0]$	$[\$300, New, \$100](S_1)$	$[1,1,0]$
$[\$300, New, AnyValue]$	$[5,3,0]$	$[\$200, Used/GoodCondition, \$60](S_2)$	$[0,0,0]$
$[\$300, New, AnyValue]$	$[5,3,0]$	$[\$60, Used/BadCondition, \$10](S_3)$	$[0,0,0]$

By multiplying each matching score with the corresponding weight and calculating the total, we obtain the following results,

Weights	*Matching Score*	*Weighted Scores*	Total
$[5,3,0]$	$[1,1,0](S_1)$	$[5,3,0]$	$8(S_1)$
$[5,3,0]$	$[0,0,0](S_2)$	$[0,0,0]$	$0(S_2)$
$[5,3,0]$	$[0,0,0](S_3)$	$[0,0,0]$	$0(S_3)$

So, we conclude that S_1 matches Q because it receives the highest *Matching* score. However, this is a trivial case that is unlikely to happen in real life. For example, consider another *Query* definition $Q = [\ \$150,\ AnyValue,\ \$65]$. Also, assume the weights are defined as $W = [5,0,3]$. In this case, there is no full match. Particularly, the value $150 in Q does not match any of the corresponding features in the available *Services*. Similarly, the value $65 in Q does not match any of its counterparts in the available *Services*. By applying the same Matching Rule, we find the following results,

Weights	*Matching Score*	*Weighted Scores*	Total
$[5,0,3]$	$[0,0,0](S_1)$	$[0,0,0]$	$0(S_1)$
$[5,0,3]$	$[0,0,0](S_2)$	$[0,0,0]$	$0(S_2)$
$[5,0,3]$	$[0,0,0](S_3)$	$[0,0,0]$	$0(S_3)$

It is not surprising that all results are $Zeros$. This is because the Matching Rule has determined that all the available *Services* are not relevant to the defined *Query*. This case is more common in practice. As shown in this example, *Matching* provides no information as to which *Service* is better. To deal with similar cases where *Matching* is not effective we introduce *Ranking* concept.

2.1.5 Ranking

Ranking is a process to *rank Services* based upon the *closeness* of a *Service* with respect to a given *Query*. The difference between *Matching* and *Ranking* is that *Matching* rewards a *Service* feature only if it exactly matches the corresponding *Query* feature, whereas *Ranking* rewards every feature of a *Service* based upon its *closeness* to the corresponding *Query* feature. Similar to *Matching*, we also perform ranking implicitly in real life. In many cases, we prefer a *Service* over another although none of the available options match our requirements completely. Nevertheless, we make our decision based on what we think is the best trade-off. In fact, we even say some statements that reflect our implicit ranking such as *"It is not the best choice, but good for its price"*, which means that we gave up some quality for a cheaper price. This also means that there was no option that meets all the desired requirements. In online *Services*, *Ranking* process cannot be performed intuitively. We need a precise algorithm for ranking. In *Ranking* algorithms, *Ranking Score (rank)* is assigned to each alternative based on a specific rule or measurement. Thus, each alternative is ordered by its *rank*, such that the highest ranked *Service* reflects higher relevance or closeness to consumer requirement. Thus, *Ranking* considers partial-matches and uses rules for measures for closeness.

Ranking algorithm must be designed with great care in order that the result produced by the algorithm reflects our intuitive expectations. The following example, similar to the one given in *Matching* section, illustrates that some ranking algorithms may mislead. We consider four *Services* to be ranked for a given *Query* vector and a weight vector.

$$S_1 = [\$300, New, \$100]$$

$$S_2 = [\$200, Used/GoodCondition, \$60]$$

$$S_3 = [\$60, Used/BadCondition, \$10]$$

$$S_4 = [\$145, New, \$105]$$

$$Q = [\ \$150,\ AnyValue,\ \$100]$$

$$W = [5,\ 0,\ 3]$$

where,

S_1, S_2, S_3, S_4 are the available *Services* in the market.

Q is the consumer *Query*.

W is the consumer level of importance associated with the *Query*.

Assume that our ranking rule is defined as follows

$$RankingRule = \begin{cases} 5 & \text{if } Q_i = S_i \\ 3 & \text{if } |Q_i - S_i| \leq \frac{Q_i}{2} \;\&\; Q_i \neq S_i \\ 0 & otherwise \end{cases}$$

By applying the Ranking Rule on each feature of the *Query* and the available *Services*, we obtain the following results,

Weights	*Ranking Score*	*Weighted Scores*	Total
$[5,0,3]$	$[0,0,5](S_1)$	$[0,0,15]$	$15(S_1)$
$[5,0,3]$	$[3,0,3](S_2)$	$[15,0,9]$	$24(S_2)$
$[5,0,3]$	$[0,0,0](S_3)$	$[0,0,0]$	$0(S_3)$
$[5,0,3]$	$[3,0,3](S_4)$	$[15,0,9]$	$24(S_4)$

By looking at the total ranks for all *Services*, we find that *Services* S_2 and S_4 received the maximum rank, S_3 received the minimum rank, and S_1 received an intermediate rank. However, the options in S_2 and S_4 are different. For a consumer service S_4 is much closer to Q than S_2. This becomes evident by looking at the following table in which the deviation between the values of *Services* and the *Query* are shown.

Feature	Q	$\|S_{2i} - Q_i\|$	$\|S_{4i} - Q_i\|$
$price$	150	50	5
$carStatus$	–	–	–
$depositPrice$	100	40	5

Since the difference between Q and S_4 is less than the difference between Q and S_2, we can say that S_4 is closer. Nevertheless, the ranking algorithm ranked S_2 and S_4 equally. The ranking method not only does not help the consumer to choose the *Service* closest to the request but it misleads the consumer by assigning similar ranks to different *Services* as if they provide the same level of quality. Therefore, we say that this Ranking Method is *unfair*. In the following section we explain *fair Ranking* and motivate the necessity to show convincingly that an algorithm is fair.

2.2 Requirements for a Fair Ranking Algorithm

Fairness is a loose term that has many perspectives. Therefore, we define fairness from two specific perspectives; from consumer perspective and from algorithm perspective. We discuss both views in the following subsections.

2.2.1 Consumer Perspective

A fair ranking algorithm from a consumer point of view is generally an algorithm that can produce results based only on the consumer's requirements, without any influence by other factors inherent to the algorithm or the system. Also, it is an algorithm that provides the consumer the closest *Service* to a *Query*. Specifically, a fair ranking algorithm is an algorithm that embraces the following characteristics.

- Allows consumers to look for *better* values: This means that the algorithm can find better values than the ones defined in the user request if the user demands it. For example, when a consumer defines a price a ranking algorithm that supports better values will find cheapest deals and provide them first. In a cheapest deal the price may not be closest to what was specified in a *Query*, but it is better for the consumer.

- Allow ranking based on *numerical and non-numerical values*: The will allow consumer to be able to use literal (textual) or numerical features in a *Service*. This reduces some of the restrictions on the consumer in finding the *Service* that best suits the requirement.

- Performs ranking Online (On-Spot) and without making any *assumptions* regarding consumer's requirement: Specifically, in addition to requirements consumers will have level of importance, and other options. Some ranking algorithms [SKR99] include offline analysis and consider other system-related criteria that are not related to the consumer requirements. This kind of analysis may result in ranking some *Services* higher not because they are close to consumer's requirement but because they meet some other system-related criteria. As a result, consumers do not receive a list that is characterized by only their requirements. The fair ranking algorithm that we discuss later will consider only consumer inputs and do not make other input or assumptions. This helps consumers to find *Services* tailored on their requirements as opposed to the output from other algorithms that make algorithmic assumptions and anticipatory consumers needs.

- Provide options to consumers to *manipulate results*: This helps consumers to see the results from different perspectives. Thus, they can choose the best trade-offs or best deals. For instance, consumers can find answers for questions like "what is the cheapest price for a best quality?, what is the cheapest price for specific rate?", and so on.

In addition to the above characteristics that are necessary for fairness, we include *efficiency* and *user-friendliness* as two desirable characteristics for our fairness algorithm. The reason for including efficiency is that consumers hate to wait. In online businesses, one of the greatest enemies to business success is slow ranking or slow operations. In fact, waiting for too long often causes a loss for both consumers and *Service* providers. This is because consumers end up not selecting any *Service* as a result of the slow process. The reason to include user-friendliness is to enable consumers build queries without taking too much time to understand how to build a request. Also, building requests should be simple regardless the number of required features and the integrated options attached to it.

2.2.2 Algorithmic Perspective

From the algorithmic perspective a fair ranking should be consistent, flexible and timely. Below is a list of characteristics for an algorithmic perspective of fairness.

- Considers different *semantics*: It is necessary to support the idea of finding *better values* purely from an algorithmic view. To be able to find better values, the algorithm should have a definition of what is a "better value"? Is a higher value better or is a lower value better? The answer depends upon the semantics of the features being compared. For some features, such as reliability, quality, and performance, ‘higher’ is better. For some features such as, cost and weight, ‘lower’ is better. For features that are described textually or by logical values (true/false) only exact match is acceptable.

- Produces *normalized* results: Normalization means producing the same range of numbers for all range of inputs. However, the specific definition of normalization is irrelevant in this view. In this level, normalized results implies that the numbers produced by the algorithm have a maximum value. Thus, regardless of the input, the output produced is always up to a maximum number. Knowing the maximum values produced by the algorithm make it more maintainable and manageable in the sense that it allows investigating the accuracy and the correctness of the results. Arbitrarily unknown results are hard to track and difficult to examine. In addition, knowing a maximum boundary helps in other subsequent processes that might use these numbers as input.

- Remains *consistent*: The algorithm produces the same result for the same input at any given time. That is, the environment in which the algorithm is executed should not have an impact on the generated results. This is because as mentioned earlier, we consider the fair ranking algorithm to be purely dependant on the consumer’s requirement.

- Accepts input in *any range* (from 0 to ∞): Some ranking algorithms [Ibr12] perform ranking only on a specific range. This puts restrictions on consumers in the sense that they cannot look for any values.

- Has no *limitation* on number of features included: Some algorithms [MMM12] accept only a specific number of features. This is because the number of features increase the complexity of the calculations to produce the results. However, we define a fair ranking algorithm as an algorithm that offers freedom of choice regarding number of features. Thus, a consumer can include as many features as are necessary.

- Provides different *options for results manipulation*: The ranking algorithms are exposed to many inputs and different cases. For example, when two features are of the same importance to the consumer, and one is the best in a *Service* and the other is the best in another *Service*, which *Service* should the ranking algorithm rank higher? In this case, there is no guarantee that a specific response would be the best for the consumer. To overcome this, a fair ranking algorithm should allow consumers to have other options that can force some changes on the ranking method and generate different results. As a result, a consumer can view the results from different perspectives.

- Is built on a *simple concept*: This requirement is important to be able to integrate all the above mentioned requirements into the ranking method while maintaining the performance. Complex methods are difficult to comprehend and, thus, harder to change.

2.3 Summary

In this chapter, we introduced a general view on the actions we perform when we seek a *Service* in real life. Then, we explained how we pass through similar experience when we seek online *Services*. After that, we introduced the basic terminologies and concepts used in online business. Also, we discussed, through examples, the associated difficulties when seeking online services. There, we illustrated the need for a *Fair Ranking Algorithm*. We followed that by providing the definition of *Fair Ranking Algorithm*. Finally, we defined the requirements needed to perform fair ranking from two different perspectives; consumer perspective and algorithm perspective. We illustrated the basic concepts with simple examples.

Chapter 3

Literature Review

In this chapter, we review the published solutions for multi-featured *Service* ranking. We first review multi-featured ranking algorithms, categorized into (1) ranking based on *Recommender Systems*, (2) ranking based on *Multi-Criteria Decision Making*, and (3) ranking based on *Similarity Measure*. For each category, we provide a discussion on what the method is, how it works and why it is not a suitable solution for our problem. Then, we present a brief explanation and analysis of the current common solutions for the multi-featured ranking problem. However, there does not exist much work done in this category. So, we narrow down our study on *Similarity Measures*. We introduce a number of *Similarity Measures* and evaluate them in terms of evaluation criteria introduced in Chapter 2.

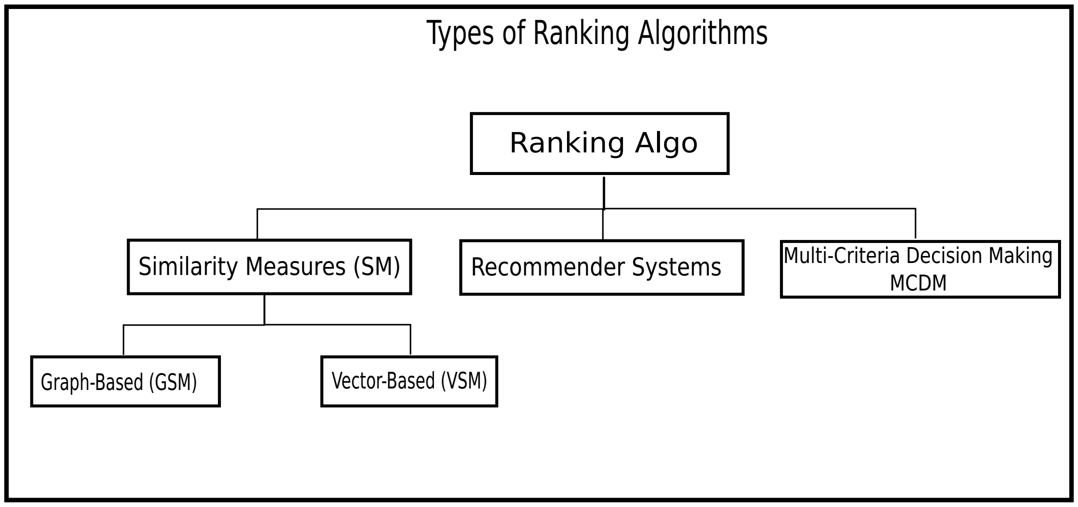


Figure 1: Different categories of existing ranking algorithms

3.1 Ranking Algorithms

In this section, we briefly explain the published ranking algorithms, by putting them into four categories. Algorithms in each category follow one specific approach. The categories are (1) Ranking Based on Vector-Based Similarity Measures, (2) Ranking Based on Graph-Based Similarity Measures, (3) Ranking Based on Recommender Systems, and (4) Ranking Based on Multi-Criteria Decision Making, as depicted in Figure 1. For each approach, we provide a discussion regarding the functionality of the method, introduce the advantages of this method and finally evaluate the method in terms of *fair ranking algorithm* requirements introduced in Chapter 2. At the end of this section, we include an overall evaluation which compares all the mentioned methods. In addition, we introduce an analysis of how current *Service* provisioning websites adopt these methods to solve multi-features ranking problem.

3.1.1 Ranking Based on Vector-Based Similarity Measures

Similarity Measure (SM) is a method that calculates the degree of *closeness* (similarity) between two items. *Similarity Measures* can be one of two types; graph-based and vector-based. A detailed on *Similarity Measures* is presented in the next section. In this section, we discuss

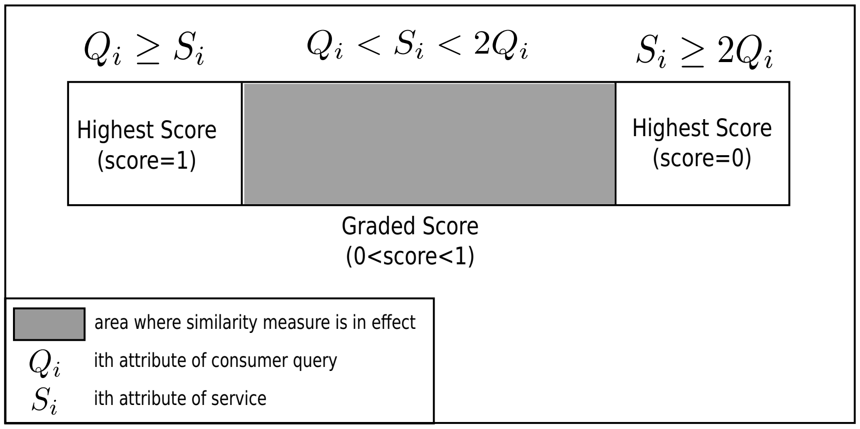


Figure 2: Ranking Algorithm Based on Similarity Measure

the ranking algorithms built on *Similarity Measure* concepts where the main scoring system employed in the ranking algorithm is based on *Similarity Measure*. Most of the published ranking algorithms fall under this category. Although there are many ranking algorithms based on *Similarity Measures*, to the best of our knowledge there exists only one ranking algorithm [Ibr12] that is proposed for multi-featured *Services*. Since our focus in this thesis is on ranking multi-featured *Services* we limit our review to the multi-featured *Service* ranking algorithm of Ibrahim [Ibr12].

The vector-based ranking algorithm [Ibr12] is a normalized ranking algorithm that produces results in the range $(0, 1)$. It only considers ranking the numerical features of *Services*. Also, this method can be applied to ranking complex or composed *Services*. The measure for ranking is defined by the following formula.

$$RankingMeasure = \begin{cases} 1 & Q_i \geq S_i \\ 2 - \frac{S_i}{Q_i} & Q_i < S_i < 2Q_i \\ 0 & S_i \geq 2Q_i \end{cases} \tag{1}$$

where, Q_i represents the i^{th} attribute of the consumer *Query*, and S_i represents the corresponding i^{th} attribute of a *Service*.

Although this algorithm is based on a simple and intuitive measure, it fails to rank *Services* fairly. That is, by looking closely at the algorithm, we find that all values of attributes of *Services* that are less than or equal to the value of attributes defined in *Query* receive the highest score. This indicates that the algorithm does not consider “better values” than the one provided in the requirement. Also, it always considers a *Service* value that is greater than the value in the *Query* to be worse. This means that it’s semantic suits only certain domains, such as cost. Moreover, the algorithm considers *Service* attributes that are twice the *Query* attributes as irrelevant by assigning them all the lowest ranking score. This means that the algorithm is useful only in a specific range of input, namely for *Service* values that lie in the range $[Q_i, 2Q_i]$, as depicted in Figure 2.

3.1.2 Ranking Algorithms Based on Graph-based Similarity Measures

Objects that encapsulate more than one feature can be represented as graphs [BM08]. A vertex in the graph represents a feature of the object and an edge represents a relationship between two features of the object. Figure 3 is an object with four features and four relationships. A *Graph-Based Similarity Measure(GSM)* is a measure that finds the degree of similarity between two given graphs. *Ranking Based on GSM(RGSM)* is a ranking algorithm that depends on a GSM to produce ranks.

RGSM models are mainly used in Pattern Recognition [DX08] [Mih04], and Image processing [RB09] applications. In these applications the ability to produce accurate results, investigate relationships between attributes and work with objects with varying number of features are more important than performance and simplicity. In fact, GSMs are known for their ability to build relations between nodes. Also, GSMs models function with binary and non-binary objects and they can produces accurate results with objects of different lengths. However, GSM models have a limited mathematical support [Mih04]. In fact, this is one of the major reasons why most of the ranking algorithms are not based on graphs. Also, GSM models are complex

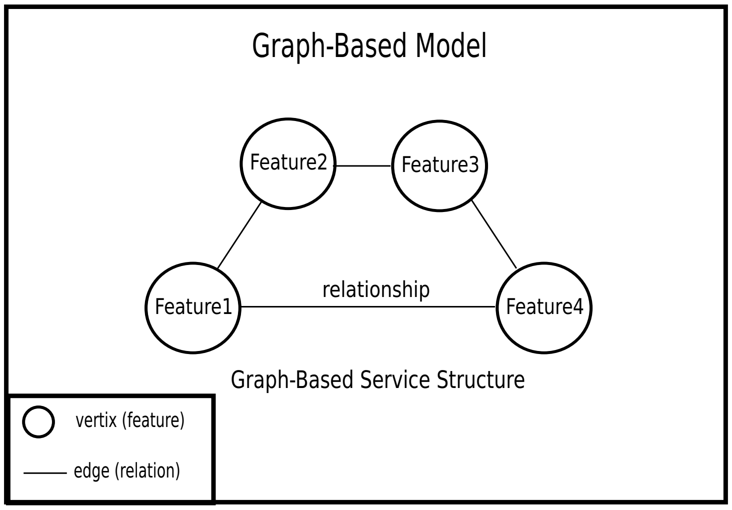


Figure 3: Structure of a service in graph-based model

and run in exponential time. This means that their performance is not efficient in terms of generating fast response to requests.

In *Service* ranking, we need a numerical measure and hence we need a model that is wealthy in terms of mathematical support. We need operations like summing two objects which are not possible with GSM. Also, unlike in Image Processing applications, we do not need to have internal relationships between the attributes of a *Service*. For instance, in a car rental *Services*, there is no need for a relationship between a car model and a car colour. Collectively, they are just features that describe the product or the *Service*. In addition, we need a fast and simple method that can handle Gigabytes of *Services* in a timely fashion which cannot be offered by GSM models. For these reasons, GSM cannot be a solution for our ranking problem.

3.1.3 Ranking based on Recommender Systems

Recommender Systems are engines that rank the objects or items based on consumer ratings and feedback. Some systems also consider the user's previous behaviour or profile [SKR99] [OH11]. It is usually an offline mechanism as it comprises heavy-load processes. That is, the recommendations and user profile are periodically collected, analysed and finally aggregated to a number

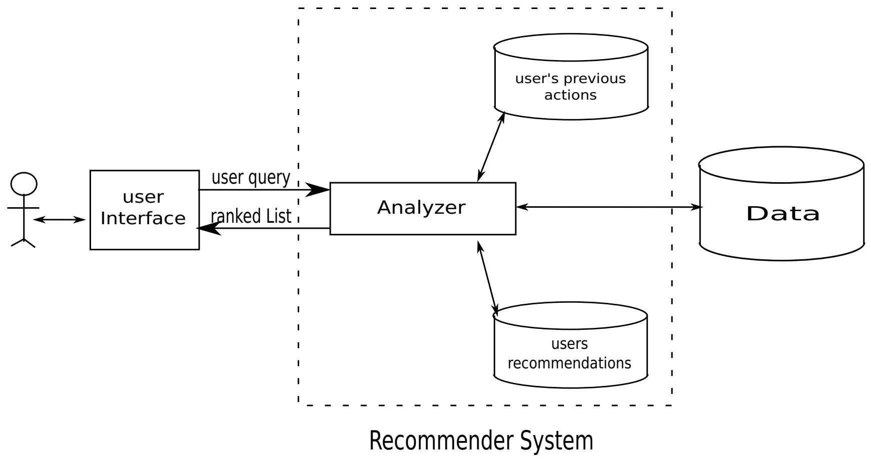


Figure 4: General Structure of Recommender Systems

that reflects a rank to particular object. Then, the objects are ordered based on the aggregated score collected at recommendations calculation periods. Therefore, a user is only required to enter a textual descriptive information regarding a specific object, such as object name, to obtain the desired object. The general structure of recommender systems is illustrated in Figure 4

The motivation for using recommendation systems for ranking can be threefold. First, it simplifies *Query* construction process for users. Second, it includes some intelligent processing in the sense that the system can anticipate what the users desire and provide it before they ask for it. Third, it benefits users and providers by directing the right *Services* to the right users.

Although the simplification of building queries is achieved, achieving the other two goals is not certain. In fact, due to the complication involved with the recommendation mechanism, problems like the harry potter problem [AT05] arose. In this problem, due to the pervasiveness of the film "Harry Potter" it was recommended at all times to all users, even for cases where the searched topic was irrelevant to the film's content. As an attempt to solve similar problems, some studies [Han09] have proposed solutions like grouping users based on many criteria such as age and interest to preserve the relevance of the recommended items. Also, trusting the validity of recommendations is another concern regarding this ranking approach. From trusting perspective, false and meaningless feed-backs are serious problems. False feedback is a feedback from a dishonest user who intend to maliciously degrade the rating of a certain provider, while meaningless feedback is an irrelevant feedback that was not meant to be for a particular product or a feedback that comes from an irrelevant user who's not interested in this particular category of products. For instance, a false feedback can be generated by a user who is loyal to another provider, while a meaningless feedback can come from a user whose interest does not match the category of the *Service* or product. Some solutions have been proposed to solve trusting issues [ZWQZ11]. They propose a framework that employs trust management models that use some methods to verify and authenticate user credentials and authorizations.

To sum up, because our goal is to provide a user-dependant ranking algorithm that can provide response on demand and in a timely manner, the recommendation systems cannot be our solution. Specifically, recommendation systems do not fit our fair ranking requirements for the following reasons. First, they lack the accuracy needed to satisfy specific user requirement, thus they are not user-dependant only. This is because they are based on anticipation rather than specific *Query* built by user. Second, they are offline methods, require tracking historical data, and consequently may not meet timeliness. This is against one of the main goals of our ranking algorithm which we claim to be timely. Third, they involve complicated, heavy-load processes and still face unsolved difficulties. Fourth, they are subjective and inconsistent processes that may vary from time to time. Therefore, recommender systems are not useful for fair ranking.

3.1.4 Multi Criteria Decision Making

Multi-Criteria Decision Making(MCDM) is considered a sub-discipline of Operations Research (OR) [Kah08]. MCDM is basically concerned with selection problems. These problems can be either fuzzy and subjective or numerical and objective. Many problems that can only be stated fuzzily, such as finding partners for marriage, making friends, and some that require making business decisions from anticipatory or observed information, are solved by MCDM method.

The goal of MCDM is to turn these problems into models based on numerical values and deal with them mathematically to arrive at a decision with the best trade-offs. Similar to ranking, MCDM is to produce scores for each alternative indicating their preference.

Because MCDM is to solve selection problems, some may argue that it is not relevant to ranking. It is true that in ranking no selection need to be made, because we can reorder based on relevance to input. However, we consider ranking and MCDM related for the following reasons. Although MCDM is used to make a decision, it does so based on some scores that it computes for each alternative which is similar to ranking. This means that we can use MCDM to rank. In many real-life applications selection of *Services* is based on ranking. For example, according to the empirical study reported in [Joa02, PHJ+07], 75% of Google users never scroll past the first page. Also, based on the report [GJG04, LHB+08] users usually look at the five top results in the list. This means, improper ranking may lead to improper selection or even non-selection, which happens when users do not care to select any alternative. For these reasons, it is legitimate to consider ranking as MCDM and vice versa.

MCDM consists of two main types. These are Multiple-criteria evaluation problems and Multiple-criteria design problems. In the former, the alternatives are either finite and known or predefined at the beginning of the process [Kah08]. This is similar to the situation for *Service* ranking. In the latter the alternatives are not defined and solutions might be infinite [Tek06]. So we consider only solutions to MCDM evaluation problems as suitable for ranking *Services*. Analytical Hierarchical Process (AHP) [Saa08] is a MCDM method that has been used to rank *Services* of the cloud [GVB12]. In this work, Service Measurement Index (SMI) defines a set of business-relevant Key Performance Indicators (KPIs) that provide a specific criteria for comparing business services. Thus, AHP method is applied on a pre-defined criteria and compare it given a specific user *Query*. The advantages and disadvantages of AHP are discussed [MMM12] [GHTH11]. These are summarized below.

Advantages of AHP

- The method is flexible and has the ability to check inconsistency between preferences. [1]
- It has the ability to rank objective and subjective data of a hierarchical nature.
- It decomposes the big comparison between *Query* and alternatives into a number of pairwise comparisons. Thus the larger problem is decomposed into smaller problems and their solutions are combined to get a solution to the original problem.
- It is the most reliable MCDM method.

Disadvantages of AHP

- It has the potential for *Rank reversals,* which means that the method produces unexpected inconsistent changes in the results when a change is performed on the alternatives set. That is, it is not a stable method that can be used for fair ranking.
- It allows compensation, which happens when good features of a *Service* hides the bad features. This is caused by the additive aggregation method adopted by the AHP method.
- It is a very complex and lengthy process, especially as the number of alternatives increase. This is because AHP decomposes the ranking into $(n(n-1)/2) * (m(m-1)/2)$ (where n is number of criteria in a query and m is the number of alternatives) pairwise comparisons.
- For each pairwise comparison AHP requires users to input their preferences based on $(1-9)$ scale. However, users may find it difficult to choose the appropriate number.

In short, AHP cannot be expected to produce a fair solution, for the following reasons. First, it requires intensive calculations of high complexity as the number of attributes increases. This implies that we have to impose some limitation on the number of features associated with

[1]Inconsistency happens when the preferences are inconsistent. For example, if a is preferred over b, and b is preferred over c, then a should be preferred over c. otherwise, the preference is inconsistent.

objects. Second, users may fail to understand the impact of each number in the scale (1,9) defined for AHP. This misunderstanding can cause an inconsistent choice by the user. Third, the inconsistency problem associated with rank reversals, albeit the attempts to solve it, remains an open issue. As a result, AHP is not the best option for building a timely, simple, user-dependent and multi-featured ranking algorithm. The example shown below is to bring out the complexity of AHP method and to support our view that much simpler and faster algorithms might achieve the same result as AHP.

Example 2. *In this example, we solve a ranking problem using the AHP method, and the vector-based ranking algorithm introduced in [Ibr12]. This example highlights the difference between the complexity of AHP and the vector-based algorithms, although they achieve the same result.*

Problem Statement:

A car is described by the three criteria price, shipping_time and shipping_cost. It is required to choose the best car for a given query $Q = [\$30000, 25days, \$1000]$, *given the three alternatives* a_1, a_2, a_3.

$$a_1 = [\$35000, 20days, \$1500]$$

$$a_2 = [\$40000, 30days, \$1200]$$

$$a_3 = [\$50000, 40days, \$900]$$

Figure 5 graphically depicts the number of pairwise comparisons associated with three Services *with three features.*

Solving the problem using AHP:

We explain the nine steps involved in AHP method without going into the mathematical rationale for this approach. Such details can be found in [GVB12]. For each AHP step we explain the calculations done in that step and illustrate the calculation for the above problem.

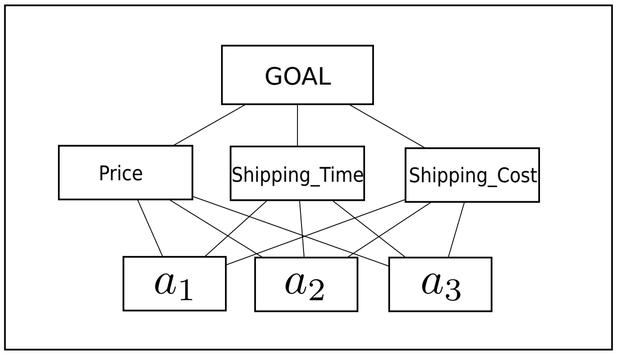
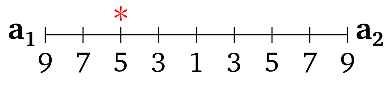


Figure 5: AHP pairwise comparisons

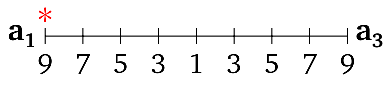
- *Step 1. Make* $(n(n-1)/2) * (m(m-1)/2)$ *pairwise comparisons and estimate the relative weights.*

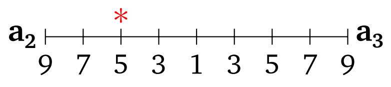
  In our problem $n = 3$ *and* $m = 3$. *We compare* a_1 *against* a_2, a_1 *against* a_3 *and finally,* a_2 *against* a_3 *for each criterion. This means that we are going to make* $3_{criteria} * 3_{alternatives} = 9$ *comparisons. Based on AHP definition, we have a* $0 \ldots 9$ *scale to estimate the preference of one alternative over the other.*

Criterion: Price

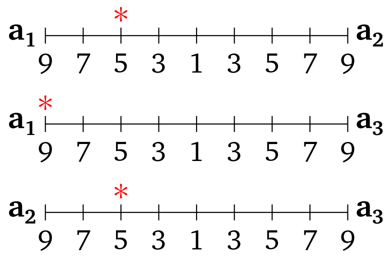


Note the star on number 5 *is the scale, which indicates that we prefer price of* a_1 *five times over* a_2. *Preferences for other comparisons are shown next.*



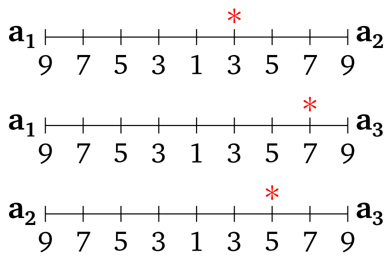


Criterion: Shipping_time(stime)



$\mathbf{a}_1$ 9 7 5 3 1 3 5 7 9 $\mathbf{a}_2$
$\mathbf{a}_1$ 9 7 5 3 1 3 5 7 9 $\mathbf{a}_3$
$\mathbf{a}_2$ 9 7 5 3 1 3 5 7 9 $\mathbf{a}_3$

Criterion: Shipping_cost(scost)



$\mathbf{a}_1$ 9 7 5 3 1 3 5 7 9 $\mathbf{a}_2$
$\mathbf{a}_1$ 9 7 5 3 1 3 5 7 9 $\mathbf{a}_3$
$\mathbf{a}_2$ 9 7 5 3 1 3 5 7 9 $\mathbf{a}_3$

- *Step 2. The Eigen vector (EV) is calculated. In this example we calculate an approximation only. For more information refer to [Tek06].*

$$\begin{array}{cccc} \textbf{price} & a_1 & a_2 & a_3 \\ a_1 & 1 & 5 & 9 \\ a_2 & \frac{1}{5} & 1 & 5 \\ a_3 & \frac{1}{9} & \frac{1}{5} & 1 \\ sum & 1.31 & 6.2 & 15 \end{array} \quad \begin{array}{cccc} \textbf{scost} & a_1 & a_2 & a_3 \\ a_1 & 1 & 5 & 9 \\ a_2 & \frac{1}{5} & 1 & 5 \\ a_3 & \frac{1}{9} & \frac{1}{5} & 1 \\ sum & 1.31 & 6.2 & 15 \end{array} \quad \begin{array}{cccc} \textbf{stime} & a_1 & a_2 & a_3 \\ a_1 & 1 & \frac{1}{3} & \frac{1}{7} \\ a_2 & 3 & 1 & \frac{1}{5} \\ a_3 & 7 & 5 & 1 \\ sum & 11 & 6.3 & 1.34 \end{array}$$

- *Step 3. The Reciprocal Matrix (RM) using EV is calculated. This is done by dividing each element of the EV matrix by the sum of its column. Thus, each element becomes normalized, and the column sum becomes* 1.

$$\begin{array}{c|ccc} \textbf{price} & a_1 & a_2 & a_3 \\ \hline a_1 & 0.76 & 0.81 & 0.60 \\ a_2 & 0.15 & 0.16 & 0.33 \\ a_3 & 0.09 & 0.03 & 0.07 \end{array} \quad \begin{array}{c|ccc} \textbf{scost} & a_1 & a_2 & a_3 \\ \hline a_1 & 0.76 & 0.81 & 0.60 \\ a_2 & 0.15 & 0.16 & 0.33 \\ a_3 & 0.09 & 0.03 & 0.07 \end{array} \quad \begin{array}{c|ccc} \textbf{stime} & a_1 & a_2 & a_3 \\ \hline a_1 & 0.09 & 0.05 & 0.11 \\ a_2 & 0.27 & 0.16 & 0.15 \\ a_3 & 0.64 & 0.79 & 0.75 \end{array}$$

- *Step 4. The priorities(PR) for each alternative on each criterion is calculated. This calculation for each alternative is basically the average of each raw of RM of that alternative.*

  Criterion:price

$$PR_{a_1} = \frac{1}{3}(0.76 + 0.81 + 0.60) = 0.733$$

$$PR_{a_2} = \frac{1}{3}(0.15 + 0.16 + 0.33) = 0.203$$

$$PR_{a_3} = \frac{1}{3}(0.09 + 0.03 + 0.07) = 0.06$$

  Criterion:stime

$$PR_{a_1} = \frac{1}{3}(0.76 + 0.81 + 0.60) = 0.733$$

$$PR_{a_2} = \frac{1}{3}(0.15 + 0.16 + 0.33) = 0.203$$

$$PR_{a_3} = \frac{1}{3}(0.09 + 0.03 + 0.07) = 0.06$$

  Criterion:scost

$$PR_{a_1} = \tfrac{1}{3}(0.09 + 0.05 + 0.11) = 0.084$$

$$PR_{a_2} = \tfrac{1}{3}(0.27 + 0.16 + 0.15) = 0.193$$

$$PR_{a_3} = \tfrac{1}{3}(0.64 + 0.79 + 0.75) = 0.724$$

- *Step 5. The Principle Eigen value(λ_{max}) is defined as*

$$\lambda_{max} = Priority_{a1} * RM_{C1} + Priority_{a2} * RM_{C2} + Priority_{a3} * RM_{C3},$$

where C refers to column sum of RM, calculated in Step 2. For our example, the calculations of λ_{max} for the three criteria are shown below.

$$price\lambda_{max} = 0.733 * 1.31 + 0.203 * 6.2 + 0.06 * 15 = 3.1$$
$$stime\lambda_{max} = 0.733 * 1.31 + 0.203 * 6.2 + 0.06 * 15 = 3.1$$
$$scost\lambda_{max} = 0.084 * 11 + 0.193 * 6.3 + 0.724 * 1.34 = 3.1$$

- *Step 6. The Consistency Index (CI), defined as*

$$CI = \frac{\lambda_{max} - 3}{2},$$

is calculated. Calculation of CI for each criterion is shown below.

$$CI_{price} = \tfrac{3.1-3}{2} = 0.05$$

$$CI_{stime} = \tfrac{3.1-3}{2} = 0.05$$

$$CI_{scost} = \tfrac{3.1-3}{2} = 0.05$$

- *Step 7. The Consistency Ratio $CR = \frac{RI}{CI}$, where RI is the Random Consistency Index [Saa83] is calculated. The calculated value of CR is considered consistent if*

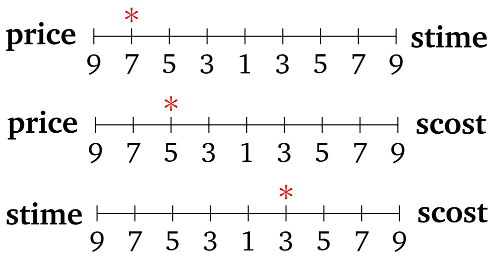
$CR >= 10\%$. In [Saa83] the Random Index RI for $n = 3$ is given as 0.58 when $n = 3$. Using it for our example, we calculate CR for each criterion,

$$CR_{price} = \frac{0.05}{0.58} = 0.086 * 100 = 8.6\% < 10\% \quad consistent$$

$$CR_{stime} = \frac{0.05}{0.58} = 0.086 * 100 = 8.6\% < 10\% \quad consistent$$

$$CR_{scost} = \frac{0.05}{0.58} = 0.086 * 100 = 8.6\% < 10\% \quad consistent$$

- *Step 8.The relative weights of each criterion are calculated. In calculating the weight of each criterion CW, we assume that price is the most important to the user, then shipping cost and finally the shipping time.*



We apply the same steps applied previously on the pairwise comparisons between each alternative. That is, we apply the steps from (step2) to (step7). Hence,

$$PR_{price} = \frac{1}{3}(0.75 + 0.64 + 0.79) = 0.72$$

$$PR_{stime} = \frac{1}{3}(0.11 + 0.09 + 0.05) = 0.08$$

$$PR_{scost} = \frac{1}{3}(0.15 + 0.27 + 0.16) = 0.19$$

Also, we find,

$\lambda_{max} = 3.11$

$CI = 0.053$

$CR = 0.095 * 100 = 9.5\% < 10\%$ *consistent.*

- *Step 9. The final rank is calculated by applying the following formula for each alternative.*

$$FinalScore_{alternative} = CW_{criterion_1} * PR_{alternative} + \cdots + CW_{criterion_n} * PR_{alternative}$$

The calculations for our example are shown below,

Criterion	*Weight*	*Alternative Ranks*
price	*0.72*	$a1_{rank} = 0.72 * 0.73 = 0.531$
		$a2_{rank} = 0.72 * 0.20 = 0.147$
		$a3_{rank} = 0.72 * 0.06 = 0.043$
stime	*0.08*	$a1_{rank} = 0.08 * 0.08 = 0.061$
		$a2_{rank} = 0.08 * 0.08 = 0.017$
		$a3_{rank} = 0.08 * 0.08 = 0.005$
scost	*0.193*	$a1_{rank} = 0.19 * 0.08 = 0.016$
		$a2_{rank} = 0.19 * 0.09 = 0.037$
		$a3_{rank} = 0.19 * 0.72 = 0.140$

To calculate the final rank for each alternative, we simply add all it's ranks. For example, for a_1 *we add the ranks for* a_1 *in price , in stime and in scost. Thus,*

Alternative	*Final Ranks*
a_1	$0.531 + 0.061 + 0.016 = 0.608$
a_2	$0.147 + 0.017 + 0.037 = 0.201$
a_3	$0.043 + 0.005 + 0.140 = 0.188$

As the results indicate, $a_1 > a_2 > a_3$ *where* $" > "$ *means better.*

Vector-based Ranking Algorithm [Ibr12]:

There are only three steps in this method.

1. *Step 1. Assume user's weights are given for each criteria, in a row vector W. Based on the algorithm's definition, a weight is a value within the range [1-5], where higher value indicates higher importance.*

2. *Step 2. calculate the property ranks matrix PR based on the following definition. The rows of PR are the features and the columns are the alternatives.*

$$PR = \begin{cases} 1 & Q \geq alternative \\ 2 - \frac{alternative}{Q} & Q < alternative < 2Q \\ 0 & alternative \geq 2Q \end{cases} \tag{2}$$

3. *Step 3. Compute the product $W \times PR$. The result is a row matrix, giving the rankings of the alternatives.*

Solution based on the above Algorithm for our Example:

1. *The weights are as in AHP method.*

$$W_{price} = 5 \qquad W_{stime} = 1 \qquad W_{scost} = 3$$

2. *We calculatePR based on Algorithm 2.*

$$PR = \begin{array}{c|ccc} & a_1 & a_2 & a_3 \\ \hline price & 0.833 & 0.667 & 0.333 \\ stime & 1 & 0.8 & 0.4 \\ scost & 0.5 & 0.8 & 1 \end{array}$$

3. *We compute $W \times PR$.*

$$FinalRank = [5, 1, 3] * \begin{bmatrix} 0.833 & 0.667 & 0.333 \\ 1 & 0.8 & 0.4 \\ 0.5 & 0.8 & 1 \end{bmatrix}$$

$$FinalRank = [4.165+1+2.5, 3.335+0.8+2.4, 1.665+0.4+3] = [6.665, 6.535, 5.065]$$

The result indicates, $a_1 > a_2 > a_3$ where " > " means better.

As the example shows, both AHP and the vector-based method of Ibrahim [Ibr12] have achieved the same results. The example highlights the complexity of AHP algorithm, which is many orders of magnitude higher when the number of features and the number of alternatives increase. Nevertheless, AHP remains a strong method that resolves fuzzy problems and make the most accurate decisions compared to other MCDM methods. However, we are not dealing with fuzzy problems, so AHP is not a preferred choice. Also, the vector-based approach of Ibrahim [Ibr12] has deficiencies which we want to overcome. So, our goal in this thesis is the investigation of simple, fast, yet considerably accurate vector-based methods for fair ranking.

3.1.5 Analysis of Current Multi-featured Ranking Solutions

In Chapter 2 we discussed the requirements of a *fair ranking algorithm* from Consumer and Algorithm perspectives. We apply these requirements to the ranking algorithms that we discussed in the previous section. We summarize our findings in Table 1 and Table 2. It is clear from this summary that none of the methods satisfy the fairness criteria.

Requirement / Methods	Vector-Based	Graph-Based	Recommender Systems	MCDM
ability to find better values	×	×	√	√
numerical/non-numerical values	×	√	√	√
online ranking without assumptions	√	√	×	√
options available to manipulate results	×	×	×	×
Fast response	√	×	×	×
allow user-friendly *Query* building	√	√	√	×

Table 1: Evaluation of Ranking Methods from Consumer Perspective

Requirement / Methods	Vector-Based	Graph-Based	Recommender Systems	MCDM
Different Semantic Support	×	×	√	√
Normalized outputs	√	×	√	√
Consistent outputs	√	√	×	×
accept input range $(0, \infty)$	×	√	×	√
No limitation on number of features	√	×	√	×
Provide options for results manipulation	×	×	×	×
built on simple concept	√	×	×	×

Table 2: Evaluation of Ranking Methods from Algorithm Perspective

All websites and E-businesses tackle the multi-featured ranking problem using a combination of two methods. One method is one of the multi-featured ranking methods introduced previously in this chapter, i.e. VSM, GSM, Recommender, or MCDM. The second method is the concept of *filtration*.

The multi-featured ranking methods employed in modern websites do not include consumer requirements. They perform ranking in two levels. The first level is performed offline before publishing the *Services*. This level includes ranking *Services* based on multi-criteria defined by the website developers and administrators. This includes factors like price, provider reputation, recommendations, number of sales, payment issued by provider to increase rank and more criteria that are not defined by consumers. Then, the second level of ranking is an online ranking where they consider the textual description of the *Service* provided by the consumer in addition to a single criterion which is the rate/rank they produced in the first level of ranking. For example, in ebay.com or Amazon.com, the user is prompted to insert a textual description of the desired *Service*/product. Then, *Services* are provided in a certain order. This order is based not only on the textual match but other implicit criteria defined by the website developers.

The second method used to have multi-featured effect is the filtration. This concept is about removing the irrelevant items from the result list. That is, the user is provided with certain tools or filters specific to each application domain. Through these filters, consumers can *filter out* the unfavourable products/*Services*. This is performed on-spot once a user clicks or choose those filters. For example, when a user wants to see items of a ranged price, there is no ranking performed. Rather, there is a filtration process that goes over all the results and filters out those products that are not within the specified price range.

Although the combination of the mentioned methods together provides the consumers with a multi-featured-like ranking, it can not be considered a fair ranking. This is due to the following reasons:

- The offline multi-featured ranking considers many factors. These factors are not dependant on the user request only. Thus, the user may receive unexpected results. This may lead in non-selection problem since the top ranked *Services* may not be the best for the user.

- Filtration does not provide consumers with other options. To clarify, in real life when we consider a specific price range, we would like to hear about those *Services* that meet our requirements first. However, we don't mind hearing about those that meet all our requirements but slightly violate the range. This gives us the exposure on wider range of *Services* and provide us with a pure freedom to make a choice. In the same view, this allows providers to have opportunity to be exposed to wider range of consumers.

Some may argue that having internal offline ranking is beneficial for website developers. In particular, it gives developers opportunity to make financial benefits by controlling the rank list. For example, developers could ask for some extra fees to locate particular providers on top of the list in particular *Service* domain. Although we don't support the idea of having any internal control in the result list, we think that fair ranking algorithm can still offer this option, with no need to offline ranking, if it has been integrated with the proper options.

Based on the evaluation shown above, we can see that there are no available methods that meet all our requirements. Nevertheless, we see that vector-based ranking are closest to our requirements. Thus, we dive into this type and try to investigate the basis of it as an attempt to come up with a method based on vectors that can meet all our requirements. The basis of vector-based ranking algorithms is called vector-based *Similarity Measure* , which will be our discussion in next section.

3.2 Similarity Measures

Similarity Measure *is a metric that is used to calculate the degree of similarity between two items.* It can be deterministic or probabilistic. SM is deterministic when its results don't change over time, and it's a probabilistic otherwise. Figure 6 shows the different types of vector-based similarity measures. Probabilistic *Similarity Measure* is used when solving fuzzy problems that depends on how the similarity is perceived at a certain time which might not be applicable in another time [AE07]. It is generally related to personal experience such as taste of product, or opinion about someone or something. Thus, it is not a useful measure within our scope of study. On the other hand, deterministic *Similarity Measure* produces the same output if given the same input at any given time based on calculations not on how it's perceived [AE07] which makes it relevant to our ranking problem. It can be applied on numbers, images, documents and any other comparable objects. However, *Similarity Measures* are mainly needed when a graded scale result is required. That is, the output expected of SM is a range of numbers indicating the degree of relevance between the compared objects. For example, SM for two documents can be 0, 5, or 10, indicating that the documents are not related, related by 50%, or identical.

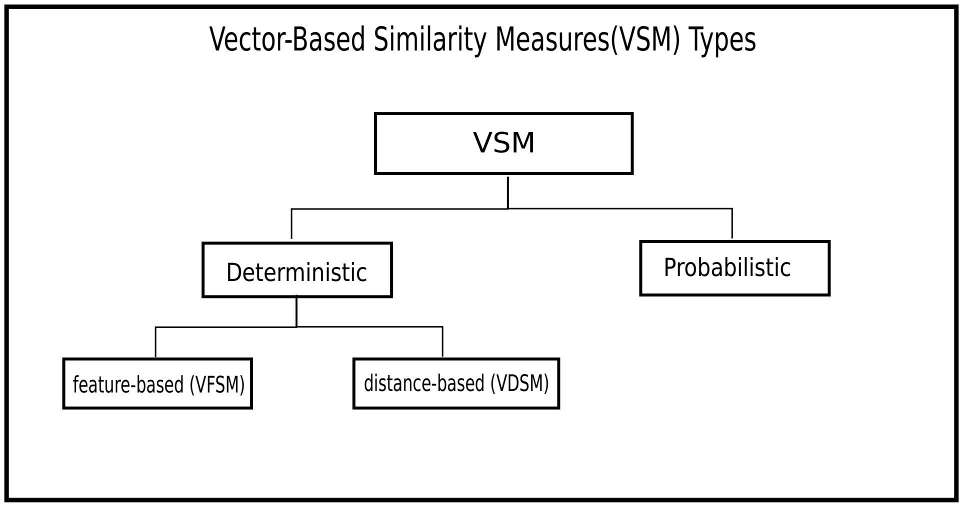


Figure 6: Different vector-based similarity measures

3.2.1 Vector-based Similarity Measure

From the review done in previous sections, we conclude that Ranking Algorithms that meet all our requirements do not exist. Also, we found out that the Vector-Based Ranking Algorithms are the closest one to our requirements. Therefore, we decided to narrow down our investigation to Vector-Based Similarity Measures (VSM) which are the basis of Vector-Based Ranking Algorithms.

Vectors can be a way of describing a multi-featured objects. A dimension in the vector is also called a feature or an attribute. However, these vectors are not the vectors introduced in mathematics and physics which are represented in vector space. Rather, the vectors we are referring to are known as feature vectors [PS09]. Basically, a feature vector is an object that contains, or is described by, multiple features [SB09]. This object and its multiple features are represented as a point in multiple dimensions in the feature space [LM98]. Nevertheless, we are still able to use vectors algebraic operations such as *Dot Products*, and *Scalar Products* on such vectors. For example, a car can be described as a feature vector, where the car's colour is a feature, its mileage is a feature and its price is a feature and so on. Thus, a VSM is a *Similarity Measure* that is used to calculate the degree of similarity between two given vectors.

The two types of VSM are Feature-based VSM (VFSM) and Distance-based VSM (VDSM) [XNJR02] [T+77]. The difference between the two types is that when comparing two objects in VFSM, the concept is to validate the equality between each attribute, and calculate the final results based on how many attributes of the objects are identical. For instance, for the vectors $Q = [a, b, c]$ and $V = [b, b, b]$, the similarity is 1. On the other hands, in VDSM the similarity of two objects is the number of attributes that differ in their vector representation. For example, for the vectors $Q = [1, 2, 3]$ and $V = [2, 3, 4]$ the similarity under VDSM is 3.

VFSM is beneficial when the only needed information is the equality between two features of a *Service*. This can work perfectly with non-numeric data types that don't provide more than true/false kind of information such as strings and boolean. Our ranking algorithm is dependant on numbers as well. In fact, one of the key requirement of our ranking algorithm is to be able to perform comparisons between features. That is, which feature is less or more than the other. However, VFSM model cannot provide such an information. This concludes that VDSM is more applicable to our case. Nevertheless, we might need to integrate the concept of VFSM in our model when we use strings and boolean data types.

VDSM has been used in numerous scientific fields such as machine learning [Aga11], bioinformatics, genealogy, chemistry, information retrieval [Sig05], computer science, mathematics, image processing and many other fields [CYT05]. Thus, number of diverse VDSM exists. Therefore, a complete review of all of them is out of reach. However, we provide a review on most of the recent ones and ones that are highly related to our topic.

3.2.2 Distance-based SMs: Current State of the Art

We have grouped the *Similarity Measures* based on the range of data they produce. That is, we have created a table for each group. *Similarity Measures* that produce unbounded results, i.e. [0,∞] Formulas listed in Table 3 produce unbound values, while formulas listed in Table 4 produced bounded values.

(A) Euclidean Dist.L_2	$d_{Euc} = \sqrt[2]{\sum_{i=1}^{d} \|S_i - Q_i\|^2}$	(B) City BlockL_1	$d_{CB} = \sum_{i=1}^{d} \|S_i - Q_i\|$
(C) Chebyshev L_∞	$d_{Cheb} = max\|S_i - Q_i\|$	(D) Minkowski L_p	$d_{Mk} = \sqrt[p]{\sum_{i=1}^{d} \|S_i - Q_i\|^s}$
(E) Kulczynski	$d_{kul} = \frac{\sum_{i=1}^{d} \|S_i - Q_i\|}{\sum_{i=1}^{d} min(S_i, Q_i)}$	(F) Gower	$d_{gow} = \frac{1}{d} \sum_{i=1}^{d} \|S_i - Q_i\|$
(G) Lorentzian	$d_{lor} = \sum_{i=1}^{d} \ln(1 + \|S_i - Q_i\|)$	(H) Inner Product	$d_{IP} = \sum_{i=1}^{d} S_i Q_i$

Table 3: Similarity Measures that produce numbers in the range $[0,\infty]$, [Cha07] [Dan80]

*where S_i is the i_{th} attribute of a *Service* and Q_i is the i_{th} attribute of user *Query*

3.2.3 Criteria of Similarity Measure Evaluation

Similarity Measure is the core of Vector-Based Ranking Algorithms. Because our goal is to come up with a *fair ranking algorithm*, our *Similarity Measure* should be built with considerations to the requirements of *fair ranking algorithms* introduced in Chapter 2. However, because the introduced requirements were concentrated on ranking level, it had the two perspectives *consumer* and *algorithm*. On the contrary, our discussion here is at a lower level, and thus, consumer perspective is irrelevant at this stage, and part of algorithm perspective will be relevant to our discussion. Nevertheless, we will perform some changes to the requirements of *fair ranking algorithm*, in order to make them focused on *Similarity Measures*. Particularly, the requirements related to “Support Semantics, Consistency of results, Input range, Restrictions on number of features, and Options for results manipulation” are considered not relevant to *Similarity Measures*. We add the two new requirements *Symmetry* and *Sensitivity to Small Differences* that are relevant to *Similarity Measure* level. Finally, we redefine the two requirements *Normalized outputs* and *Simplicity*, that were introduced previously but explained generally.

(I) Sørensen	$d_{sor} = \frac{\sum_{i=1}^{d}\lvert S_i - Q_i\rvert}{\sum_{i=1}^{d}(S_i + Q_i)}$	(J) Soergel	$d_{sg} = \frac{\sum_{i=1}^{d}\lvert S_i - Q_i\rvert}{\sum_{i=1}^{d} max(S_i, Q_i)}$
(K) Cosine	$S_{cos} = \frac{\sum_{i=1}^{d} S_i Q_i}{\sqrt{\sum_{i=1}^{d} S_i^2}\sqrt{\sum_{i=1}^{d} Q_i^2}}$	(L) Canberra	$d_{can} = \sum_{i=1}^{d}\frac{\lvert S_i - Q_i\rvert}{S_i + Q_i}$
(M) Jaccard	$d_{jac} = \frac{\sum_{i=1}^{d}(S_i - Q_i)^2}{\sum_{i=1}^{d} S_i^2 + \sum_{i=1}^{d} Q_i^2 - \sum_{i=1}^{d} S_i Q_i}$	(N) Harmonic Mean	$d_{HM} = 2\sum_{i=1}^{d}\frac{S_i Q_i}{S_i + Q_i}$
(O) Dice	$d_{dice} = \frac{\sum_{i=1}^{d}(S_i - Q_i)^2}{\sum_{i=1}^{d} S_i^2 \sum_{i=1}^{d} Q_i^2}$	(P) Angular Similarity	$S_{Ang} = 1 - \frac{\cos^{-1}(S_{cos})}{\pi}$
(Q) RC	$S_{RC} = \sum_{i=1}^{d}\frac{\lvert Q_i - S_i\rvert}{max(Q_i, S_i)}$	(R) Service Ranking	$d_{SR} = \sum_{i=1}^{d} 2 - \frac{S_i}{Q_i}$

Table 4: Similarity Measures that produce numbers in the range $[0, Constant]$, [Cha07] [BB08] [Ibr12] [JRVF09]

*where S_i, Q_i are the i_{th} attributes of *Service*,*Query* respectively. *RC* is Relative Change

Therefore, the following is a set of requirements that are relevant to *Similarity Measures* evaluation:

- *Symmetry*: With respect to *Similarity Measure* we define symmetry in this section.
- *Normalized Outputs*: With respect to *Similarity Measure* we redefined Normalized outputs in this section.
- *Sensitivity to Small Differences*: We define this concept with respect to *Similarity Measure*.
- *Simplicity*: Specific to *Similarity Measure*, redefined in this section

In the following discussion we explain these requirements, their importance for fair ranking, and provide examples.

1. *Symmetry*: Let $B = [b_1, b_2, \ldots, b_k]$, $B' = [b'_1, b'_2, \ldots, b'_k]$ be two *Services*, and $Q = [q_1, q_2, \ldots, q_k]$ be a *Query*. Let f be the function that computes the degree of similarity between every pair of components of the vectors. We say f has symmetry property if $f(b_i, q_i) = f(b'_i, q_i)$ whenever $|b_i - q_i| = |b'_i - q_i|$

   Example 3. *Let $q = 100$, $b = 10 + q = 110$, $b' = q - 10 = 90$ where q is an attribute in the user Query, b and b' are values of attributes in two different Services. A symmetric* Similarity Measure *should produce equal scores for b and b' with respect to q.*

   Symmetry is important for the following two reasons:

   - When a user expects exact match and exact matching is not possible in the available *Services*, the *Similarity Measure* with symmetry produces the rankings which are closer to user expectation.

   - To pave the road for introducing semantic preference in our ranking algorithm. During ranking, it is better to group together similar *Services*, which is achieved by *Similarity Measure* and then apply semantics within a group to discriminate the best semantically. To clarify this point, let us consider the semantics of attributes. Let the cost attribute be q in *Query*, and b,b' are attributes in two different *Services* as defined above in Example 3. Then, even though b and b' have the same difference from q, b' is better as it is less costly. Because symmetry will force b and b' to be ranked the same, we boost the score of b' by adding a small constant value to it. Because the constant is small, it won't produce imbalance in the overall ranks, and at the same time it would make b' rank higher than b. Further details on the choice of boost value are introduced inChapter 5.

2. Normalized scores: Scores are normalized, in the sense that they are adjusted to a common scale. Consequently, scores are bounded. As an example, let the attribute of a single-attribute *Query* be $q = 100$ and the attribute values of three single-attribute *Services* be $a = 90$, $b = 80$ and $c = 70$. The scores produced by the *Similarity Measure* are normalized if they are relative to the same bound. If the normalization constant is chosen to be 100, the normalized scores for a, b and c are 0.90, 0.80 and 0.70 respectively.

   This criterion is important for the following reasons:

   - To produce interpretable results. In fact, similarity measures in a ranking system maybe exposed to unbounded numbers. Theoretically, the attributes of *Services* and *Query* are lower bounded by zero but upper bounded by $+\infty$. This means that if the *Similarity Measure* does not produce normalized outputs, it will produce an unexpectedly sparse numbers which cannot be understood.
   - To minimize the influence of big numbers (scores) in the final ranking. When a *Query* has many attributes, the ranking or scoring system has to aggregate the sub-scores in some manner to compute the final score. One example of aggregation is the additive aggregation, in which all sub-scores are added to compute the final score. Having differences in the range of values of attributes will affect the fairness of the results in the sense that attributes with big values will have more influence on the total rank. By normalization of each attribute value, this influence can be eliminated.

3. Sensitive to small changes: We want a *Similarity Measure* to be able to detect small differences between *Services*. As an example, let a *Query* be described by a single attribute q, where $q = 50$, and let available *Services* also described by single-attributes b and c, where $b = 50.01$ and $c = 50.02$. A *Similarity Measure* is sensitive to small difference if it produces different scores for b and c.

This characteristic is necessary for the following reasons:

- To ensure fairness of ranking it is necessary that even a small difference in the input should make a difference in the produced ranks.
- The algorithm can be employed to a variety of application domains where a small percentage of change is crucial to correct ranking. These include trading in precious metals and dealing with the presence of chemicals in food and drugs.

4. Simple: A *Similarity Measures* should be simple to calculate and easy to apply. Mathematically simple means that it's function definition is linear. Also, it should not be prone to mathematical violations such as division by $Zero$. Practically simple means it can be applied easily. This involves that it involves minimum number of operands and operations. The simplicity is important for the following reasons:

   - To easily integrate the *Similarity Measure* with user preferences and other options.
   - To easily integrate other features of our algorithm such as semantic-awareness and different datatypes support.
   - To introduce composition and other subsequent calculations while maintaining the timeliness of the execution.

3.2.4 Evaluation Method

In this section, we explain the experimental analysis conducted on the *Similarity Measures* listed in Tables 3, 4 for the three specific properties symmetry, normalized output and sensitivity to small changes. The goal of this exercise is to test each published method for its potential for use in a fair *Service* ranking. In order to achieve this goal we created specific datasets for testing specific properties, and used Matlab [HL97] for numerical calculations, analysis, and graphical displays.

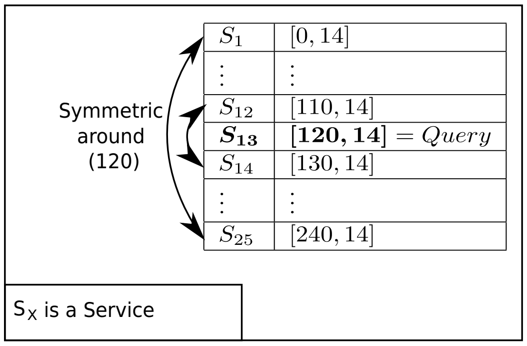
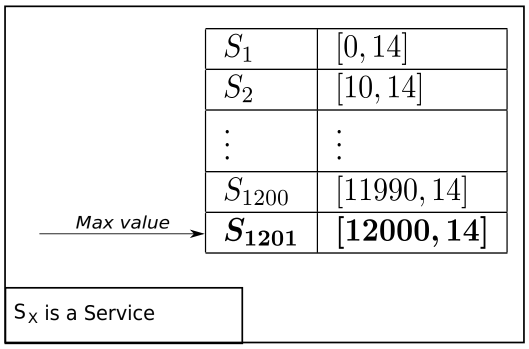


Figure 7: Dataset for testing the Symmetry property

We restricted to *Query* and *Services* with two attributes. The outcome of the experiments do not depend on the *Query*, but rather on the available *Services* and the methods that compute similarity measures, so we fixed the *Query* $Q = [120, 14]$ and varied the datasets of available *Services* for conducting the experiments.

Let $S_x = [att_1, att_2]$ be a *Service*. In general, many *Services* are created by assigning values at random to att_1, and att_2. However, it is sufficient to make an analysis for symmetry and other properties on one attribute at a time. Therefore, we can fix the value of one attribute in the *Service*, say att_2, and then vary the values of att_1 to create a dataset. There is no loss of generality in fixing the value of att_2 to 14, which is the same as the value of the second attribute in the *Query*. The reason is if we choose any other number there will be a constant (the difference between the value of att_2 and q_2) factor throughout the calculations and it will not affect the final outcome.
We use the notation $Q(1)$ to refer to the first attribute of the *Query*, and $Q(2)$ to refer to the second one. Similarly, $S_x(1)$ refer to the first attribute of the x_{th} *Service*, and $S_x(2)$ refer to the second attribute of the x_{th}.

The numbers of $Q(1)$ and $Q(2)$ were chosen randomly with no specific reasons. However, values of S_x were chosen carefully to examine specific characteristics.
In the following paragraphs, we explain more details about each characteristic:



S_1	$[0, 14]$
S_2	$[10, 14]$
$\vdots$	$\vdots$
S_{1200}	$[11990, 14]$
$\mathbf{S_{1201}}$	$\mathbf{[12000, 14]}$

Max value
S_x is a Service

Figure 8: Dataset for testing the Normalization property

1. **Testing Symmetry Property**: We created the dataset in Figure 7. We started our *Services* list with *Service* $S_1(1) = 0$. Then, we added a value of 10 to first attribute of the next *Service* in each step. Hence,

$$S_2(1) = S_1(1) + 10 = 10$$

,

$$S_3(1) = S_2(1) + 10 = 20$$

$$\vdots$$

$$S_x(1) = S_{x-1} + 10 \text{ where } x \neq 1$$

   The last *Service* in the dataset, $S_{25} = 2 \times Q(1) = 240$ which is an image to $S_1(1)$ around $Q(1)$. We call this dataset a symmetric dataset because each *Service* in the dataset has an image around Q. We compute the similarity using each one of the function defined Table 3 and Table 4. We plot the results of this study in Figure 10. In addition to the practical study, we formally examined the symmetry property for each function and included the results in Appendix A.

2. **Testing Normalized Output Property:** Figure 8 displays the dataset used to test this property. We took a large dataset because we wanted to examine whether scores are produced within specific range no matter how large is the difference between the *Query* and

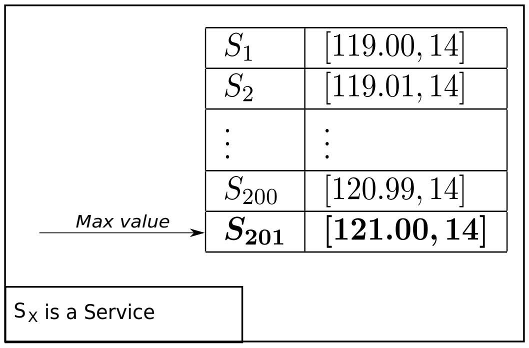


Figure 9: Dataset for testing the Sensitivity to Small Changes property

the *Service*. Our practical findings is plotted in Figure 11. Additionally, similar to Symmetry, we have supported our practical finding with a formal study on each *Similarity Measure* in Appendix B.

3. **Testing Sensitivity to Small Changes Property:** Figure 9, shows the dataset used to test this property. We made a dataset with small differences between each *Service*. Thus, we started our *Services* list with *Service* $S_1(1) = 119.00$. Then, we added a value of 0.01 to first attribute of the next *Service* in each step. Hence,

$$S_2(1) = S_1(1) + 0.01 = 119.01$$

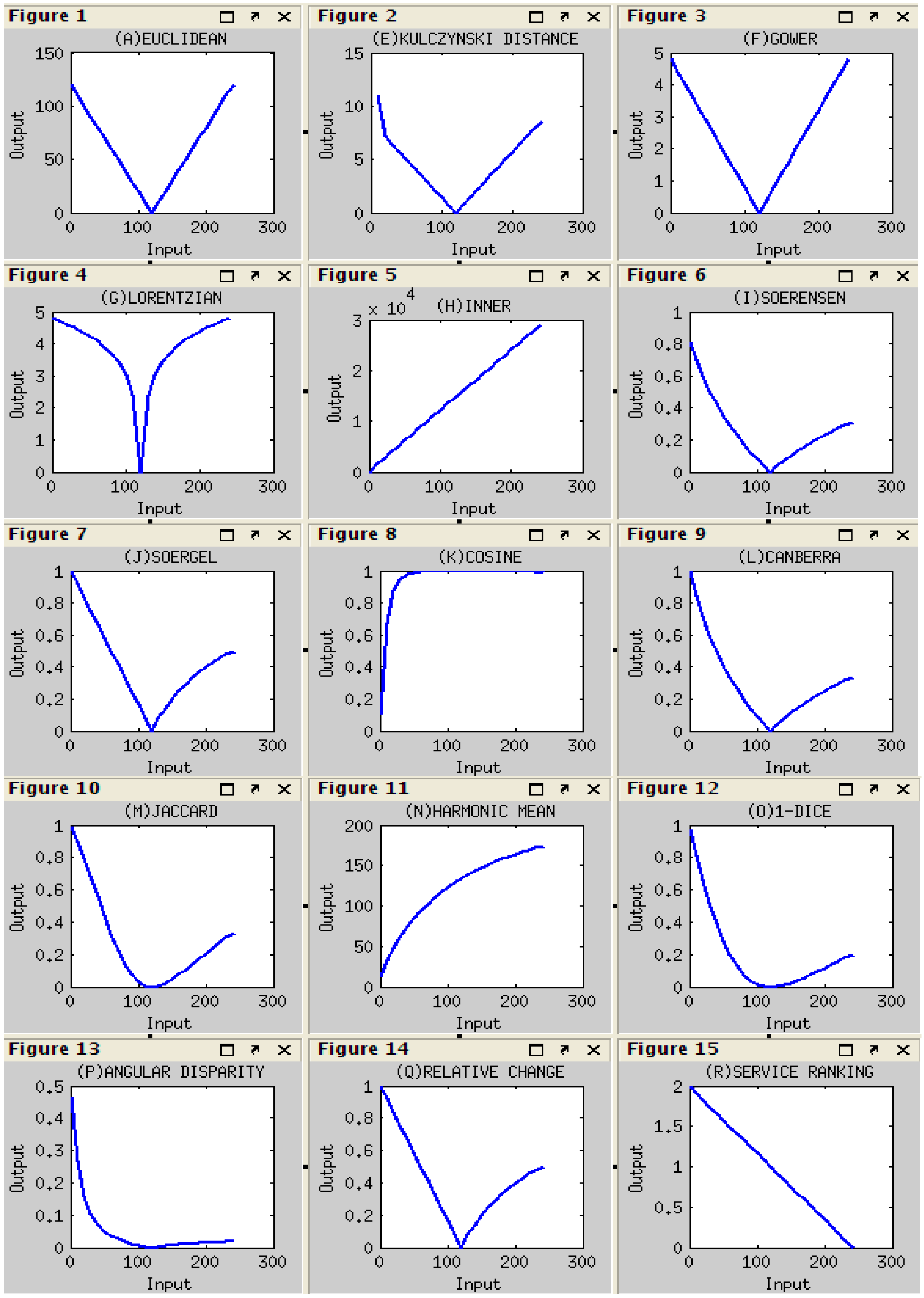
,

$$S_3(1) = S_2(1) + 0.01 = 119.02$$

$$\vdots$$

$$S_x(1) = S_{x-1} + 0.01 \text{ where } x \neq 1$$

The last *Service* in the dataset, $S_{201} = 121.00$ which is an image to $S_1(1)$ around $Q(1)$. Hence, the dataset ranged between $[119.00 - 121.00]$. We illustrate our findings regarding sensitivity in Figure 12.



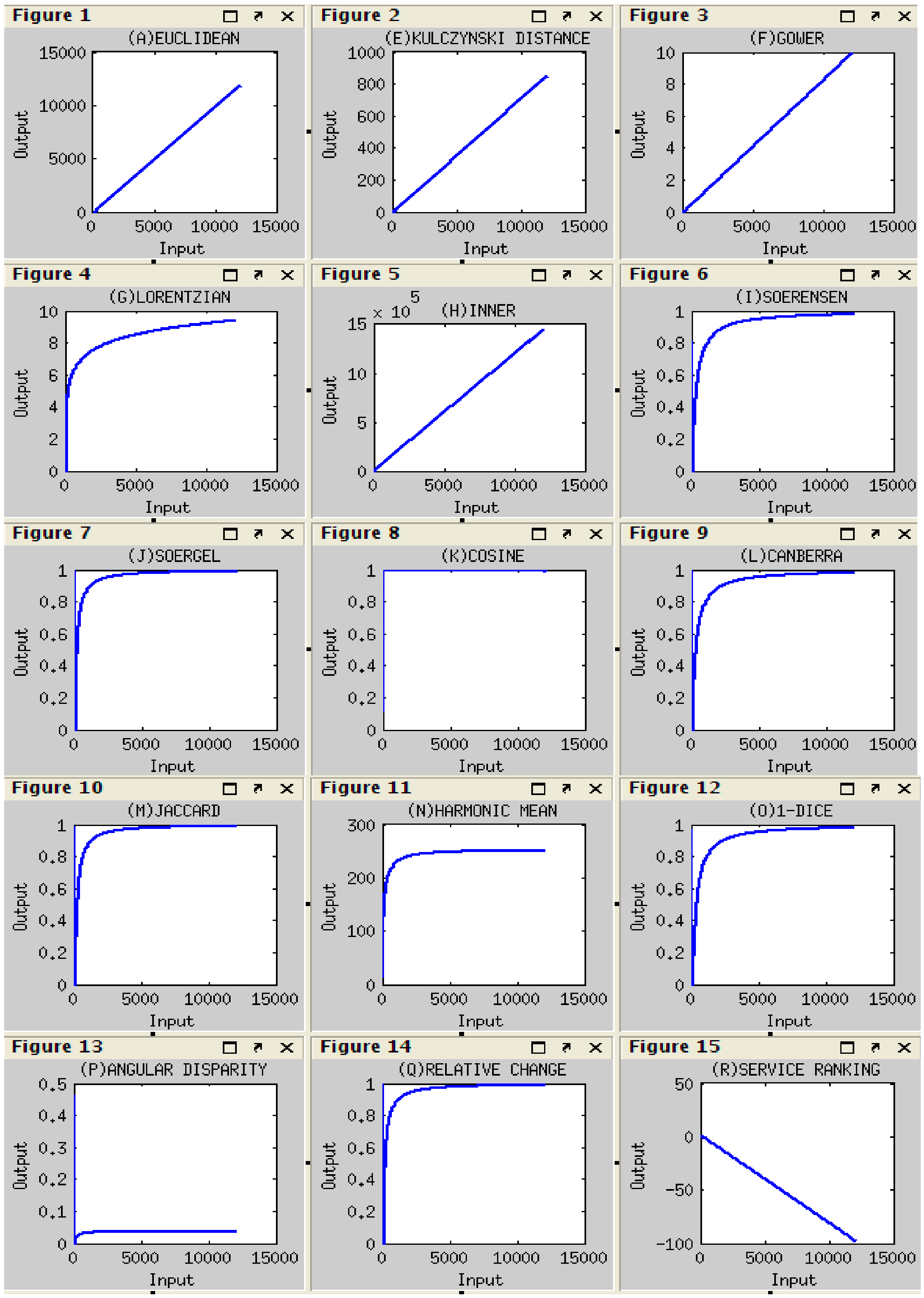
$Q(1) = 120,\ S_x(1) = [0-240],\ \#Services = 25$

Figure 10: Examining the Symmetry Feature in similarity measures

3.2.5 Observations and Analysis

Similarity Measures (A),(B),(C), and (D) in Table 3 come under Minkowski family, and their characteristics are similar. Thus we only include one of them, which is Euclidean distance. *Similarity Measures* (H) and (N) in Table 3 are not considered, as they fail to comply with any of the required characteristics. Below, we evaluate each *Similarity Measure* in terms of the requirements introduced earlier.

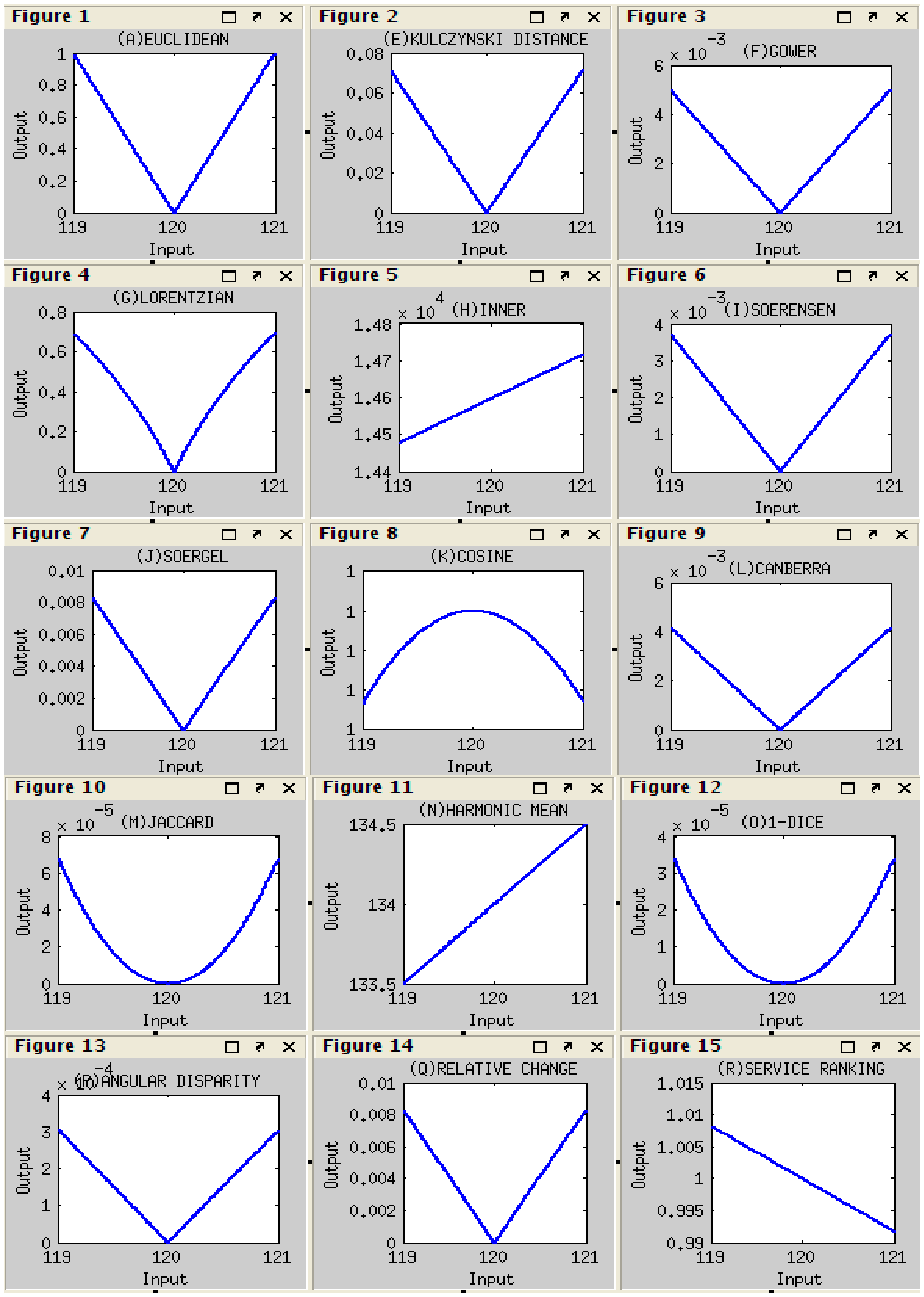
1. **Symmetry**: Based on Figure 10, *Similarity Measures* (E), (H), and (R) from Table 3 and *Similarity Measures* (I), (J), (K), (L), (M), (N), (O), (P), and (Q) from Table 4 are asymmetric. On the other hand, *Similarity Measures* (A), (F) and (G) from Table 3 are symmetric. The reason why (A),(F) and (G) are symmetric is because they are either divided by a constant like (F) or they are based on absolute difference, i.e not normalized. However, if we look at the asymmetric *Similarity Measures*, we find all of them normalized which is the main reason why they are not symmetric. That is, because they are all normalized to the values of S, Q or a sum of both, any change in the values of Q or S generates a change in the final value, even if it is a symmetric change. For example, The *Similarity Measure* (L) is normalized to |Q+S| Table 4, thus, if we consider $Q = q$, and $S_1 = q + 10$. By substitution in (L) formula we find $d_{can} = \frac{10}{2q+10}$. However, if we assume $S_2 = q - 10$, which is an image value to S_1 around Q, we find $d_{can} = \frac{10}{2q-10}$ which is not equal to $\frac{10}{2q+10}$.



$Q(1) = 120$, $S_x(1) = [0-12000]$, $\#Services$ =1201, with difference of 10 between each *Service*

Figure 11: Examining the upper bounds of the similarity measures

2. **Normalized Outputs:** By looking at Figure 11, we can identify the bounded and unbounded *Similarity Measures*. Specifically, All *Similarity Measures* in Table 3 are unbounded where all *Similarity Measures* in Table 4 are bounded. As mentioned previously, symmetry and normalization are contradictory features. The absence of one allows the presence of the other. This is explained earlier in the symmetry discussion introduced above.



$Q(1) = 120$, $S_x(1) = [119.00 - 121.00]$, *#Services* =201, with difference of 0.01 between each *Service*

Figure 12: Examining the Sensitivity to Small Changes Feature in similarity measures

3. **Sensitivity to Small Differences:** In Figure 12, we find two types of shapes. shapes that are curvy in the middle, and shapes that are pointy. This difference indicates the ability/inability of the *Similarity Measure* to capture small changes in S values. The curvy shapes are formed because the *Similarity Measure* produced similar results for different S values that differ by 0.01. In fact, the results are not exactly similar, but there are high precision differences that a regular arithmetic tool would consider them similar. Although this problem can be avoided by using special arithmetic tools that can detect high precision variations, it is not preferable solution since this complexity is created only with 0.01 difference. The pointy shapes are formed because the *Similarity Measure* produced different results with each change in S values, which is an indication of the ability to capture small differences in the input. There are only three *Similarity Measures* that are insensitive to small changes in the input data. These are *Similarity Measures* (K), (M), and (O) in Table 4. The reason behind this limitation is the fact that the results of these *Similarity Measures* are normalized to big numbers. Thus, the bigger the denominator in *Similarity Measures* the higher the precision of the change in the produced numbers and the less the ability of regular arithmetic tools to detect them. Although the curves in Figure 12 are symmetric, they don't prove the presence of this characteristic in the corresponding *Similarity Measure*. The curves are symmetric simply because the numbers are different from each other by a very small difference which make the results almost similar and hide the asymmetric nature of *Similarity Measures*. For instance, consider our *Similarity Measure* to be (I) and assume $Q = 10$ and let's have two symmetric inputs $S_1 = 9.99, S_2 = 10.01$. Hence, by substituting in *Similarity Measure* (I), we get $r_{s1} = d_{sor} = \frac{|9.99-10|}{10+9.99} = 0.00050025$, while $r_{s2} = \frac{|10.01-10|}{10+10.01} = 0.00049975$. Thus, the difference between r_{s1} and r_{s2} is 0.0000005 which cannot be detected in the graph.

Remark. Similarity Measures *(J) and (Q) in Table 4 seem to be similar but they are not. In*

Criterion / SM	(A)	(E)	(F)	(G)	(I)	(J)	(K)	(L)	(M)	(O)	(P)	(Q)	(R)
Symmetry	√	×	√	√	×	×	×	×	×	×	×	×	×
Normalized outputs	×	×	×	×	√	√	√	√	√	√	√	√	×
Sensitivity	√	√	√	√	√	√	×	√	×	×	√	√	√
Simplicity	√	√	√	×	√	√	×	√	×	×	×	√	√

Table 5: Comparison between all Similarity Measures.

essence, in (J) the division is performed once between the total of differences over the total of maximum values. However, in (Q), the division is performed for each element of S. The difference is not clear in the graphs because we assumed that the other element of the S vector is fixed. However, if we assume for instance that $S = [110, 15]$ and $Q = [120, 20]$, we find out that (J)'s result is $R_{(J)} = 0.107$ while (Q)'s result is $R_{(Q)} = 0.333$. This means that (J) produces smaller number, and details of each element of S might be lost with large set of S. One the other hand, (Q)'s result keeps more details about each element.

Table 5 summarizes the comparison between the *Similarity Measures* in Table 3 and Table 4

In summary, none of the examined *Similarity Measures* meets our requirements. However, there are six *Similarity Measures* that meet three requirements out of four. These are *Similarity Measures* (A), (F), (I), (J), (L), and (Q). The first two don't meet the normalization requirement. The rest lack the symmetry characteristic. We want to avoid the mathematical study involved with normalizing unbounded numbers. Thus, we exclude (A) and (F) from our list of options. As a consequence, we are left with *Similarity Measures* (I), (J), (L), and (Q). The *Similarity Measures* (I) and (J) are excluded because they aggregate the results of all attributes of *Query* and *Service* in one step. This makes it hard on us to integrate scalars with each element. Also, all the candidate *Similarity Measures* except (Q) have denominators as total values. This increases the value of denominator and thus increases the precision needed to recognize small differences. Unlike other candidates, Relative Change (Q) *Similarity Measure* shows a simple structure and captures highest details. Also, it only includes one number in the denominator which makes it always able to capture smaller differences. Thus, we decide to improve on it and try to customize it to meet our required characteristics.

Chapter 4

General Overview on the Proposed Solution

In this chapter, we introduce a general overview of the proposed fair ranking algorithm, the X-Algorithm. We first provide the basic mathematical background for the algorithm. Then, we explain the algorithm in the two dimensions, the front end and the back end. The front end represents the part of the algorithm that relates to end-users, where users need to define a query, preferences and make use of the options offered by the algorithm. There, we introduce, in abstract, all the options available for the user. Also, we illustrate how they can be conceptually structured altogether as one entity. After that, we introduce the back end, where the algorithm performs the actual ranking based on users' entries and requirements passed through the front end. As part of this discussion, we briefly discuss the steps to obtain the final ranking results. At the end of the chapter, we briefly summarize the information given in this chapter.

4.1 Basic Information about The X-Algorithm

The X-algorithm is a user-based, semantic-based and vector-based ranking algorithm; It is to rank *Services* in service-oriented systems. The Algorithm is called The X-Algorithm because of the "X" variable that is used to introduce semantic-awareness in the ranking algorithm. Details about this variable and how it helps to rank diverse semantics differently is introduced in next chapter. It is user-based in the sense that it produces results based on only the user's requirements. That is, it does not make any assumptions or infer any knowledge from external resources or previous behaviour such as recommender systems. The X-Algorithm is also a semantic-aware algorithm. This is because the ranks produced by the algorithm considers the semantics or type of data in addition to its value. Finally, the algorithm is a vector-based algorithm as it considers *Query*, *Services* and all other listwise variables as feature-vectors[1]. Essentially, the X-Algorithm is based on a *Similarity Measure*, called Relative Change(Q), introduced in the previous chapter.

4.2 The X-Algorithm Front-end

This section is divided into two subsections. In the first subsection, we enumerate the options offered by the X-algorithm to end-users. Also, we introduce a brief explanation for each option. In the second subsection, we illustrate, through figures, the conceptual structure of the user request, including the *Query*, *Weights*, and *Preferences*. Finally, we introduce a pseudo code for the user request.

4.2.1 Options provided to user

In this subsection, we enumerate the available consumer options that enforce changes on the results. The algorithm offers two types of options; *feature-option* and *query-option*. The feature

[1]feature-vectors are n-dimensional vectors that are used to describe a specific object [LM98, LY93]

options can be set for each feature separately. On the other hand, query-option is an option that can be set for the entire *Query*, and cannot be different for different features in the *Query*.

Feature-Options

1. Exact/Best Mode: Each feature of the *Query* can be in one of these modes. *Exact Mode* means the user is interested to find *Services* that exactly match the values in the *Query*. In contrast, *Best Mode* means the user is interested to find better values than the one defined in the *Query*. That is, in *Exact Mode* the value defined in the *Query* is considered the best value, while in best mode, the value defined in the *Query* is the minimum requirement.

2. Range of values: This option is available only for numerical features. It implies that the user chooses to enter range of values rather than one single value. An example is when the user enters a range of accepted price.

3. Essential values: This option is available for numerical and non-numerical features. It allows the user to make one feature or group of features as non-negotiable. This prompts the algorithm to first meet this/these essential feature(s) and then the other non-essential features.

4. Semantics: This option can be either system-based or user-based. In case it is to be set by the user, it is included in the user request. Through this option the user can define the meaning for better or worse to a specific feature. Thus, the features can be ranked based on the semantics. For example, if a cheaper price is better for a user, the semantic for this feature should be "Less is Better". This will prompt the algorithm to prefer the less expensive prices over the more expensive ones. Similarly, if the user defines it as "More is Better", the algorithm will prefer the more expensive prices over the less expensive ones.

Query-Options

1. AllBest option: This option is available for the entire *Query*. Once set, it prompts the algorithm to prefer the *Services* that meet all requirements for all features first. Thus, it forces the algorithm to prefer *Services* that offer meet all requirements (even if minimum requirement) for all features over ones that offer good values for all features but violate the requirement of one feature.

2. Algorithm Accuracy: This option can be either a system-based or a user-based option. In case it is considered a user-based, it is included in the user request. Thus, user can set the accuracy of the algorithm as to indicate an acceptable error rate.

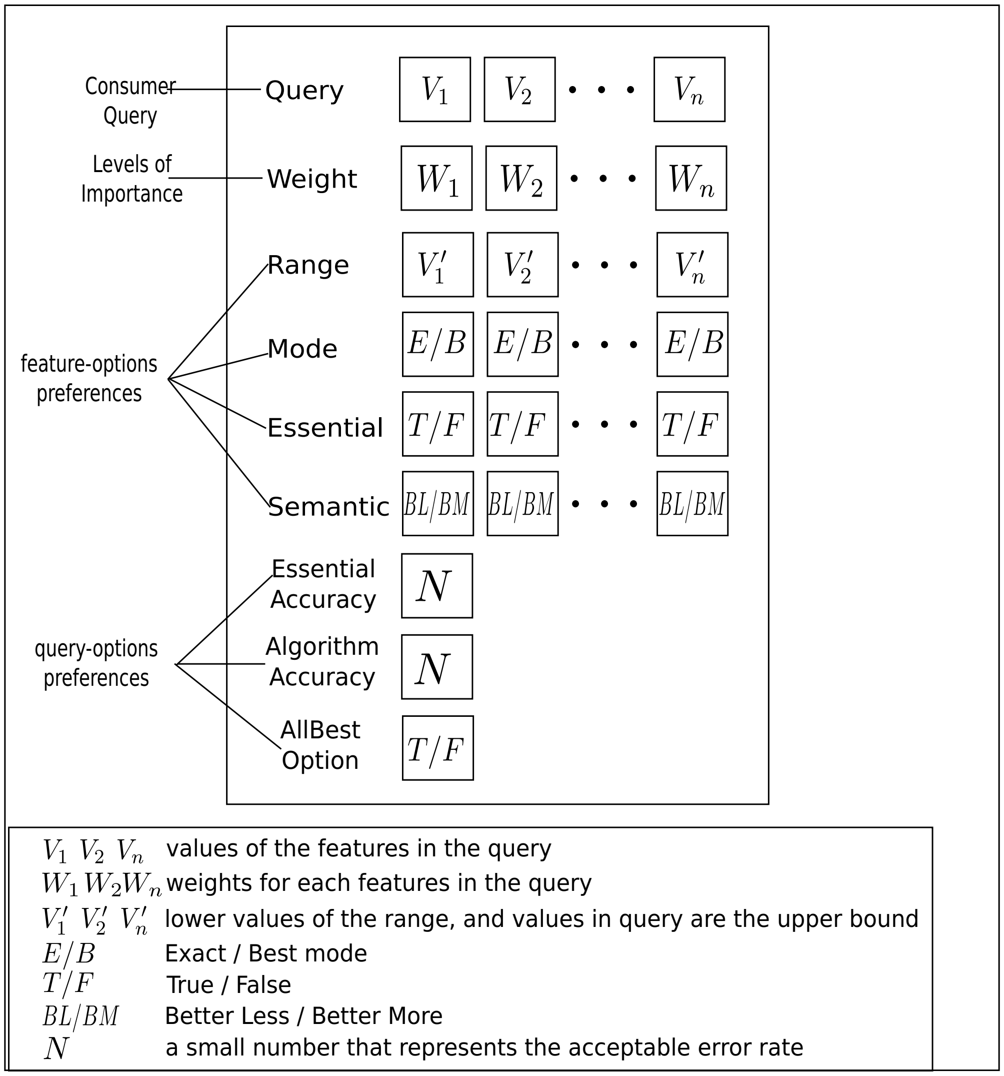
3. Essential Accuracy: Similar to algorithm accuracy, it can be based on system or users. When it is a user-based, it is included in the user request as to indicate the error rate of the essential option. Thus, this option is in effect only if essential option is "ON" for any feature.

4.2.2 Conceptual Structure of User Request

In this subsection, we provide a conceptual structure of the user request. User request consists of three entities; *Query* which includes the preferred values for each feature, *Weights* which includes level of importance for each value defined in *Query*, and *Preferences* which contains the options enumerated in the previous subsection. To combine those three entities in one object, we introduce the conceptual structure in Figure 13.

Example 4. *In this example, we provide a user request for a wireless service. Assume the wireless service has the four features: Price per month(CAD), service range(meter), Internet Speed(MB/second), Download Limit Per Month(GB). Thus, it's defined as*

$$Q = [Price(CAD), ServiceRange(m), Speed(MB/s), Download(GB/month)]$$



V_1 V_2 V_n	values of the features in the query
W_1 $W_2 W_n$	weights for each features in the query
V_1' V_2' V_n'	lower values of the range, and values in query are the upper bound
E/B	Exact / Best mode
T/F	True / False
BL/BM	Better Less / Better More
N	a small number that represents the acceptable error rate

Figure 13: The conceptual structure and Pseudo Code of the user request

Assume that the user is interested in a range of prices and prefers the price to be within the range (80\$-40\$). However, within the range the user thinks less values are better. Also, the user is interested in better(higher) Internet speed and download limit where the minimum requirement is (15 MB/second) and download limit of (120 GB/month). However, for security purposes, the user wants the exact service range specified in the Query as to not make the wireless service to be accessible from outside certain area. The user wants to see first the services that meet all requirements. Hence, the most important feature is the price, then comes the service range, then *come speed and download limit which are of the same importance.*

Assume our weights are in the range $(0,5)$ *where* 5 *means most important, and* 0 *means not important. Considering user requirements, we can translate the requirements to query, weights and preferences. Hence,*

- *Price: Options Range and Essential are used. This is because price is non-negotiable feature. Also, the semantic of price is "Better is Less" as the user wants to see less values within the range first.*
- *Service Range: Exact Mode option is used. Thus, algorithm will look for exact values or closest to one defined in the query.*
- *Speed: Best Mode option is used with the semantic "Better is More". This is because higher speed and download are better as the user specified in the requirements.*
- *Download: Best Mode option is used with the semantic "Better is More".*
- *Because user wants to see services that meet all requirements first, ShowAll option is used.*
- *For options that are not specified by users, like Mode or semantics for any feature, we use* ANY *to indicate that it does not matter for the user.*
- *In this example, we assume the accuracy is set by the system.*

Following the conceptual structure introduced earlier, and considering the definition of the user request defined above, we define the following pseudo code for the user request.

Algorithm 1: Pseudo Code of the user request object

```
begin
    Class: UserRequest
        Query= [80,10,10,120]
        Weights= [5,5,3,3]

        // feature-options preferences
        Range = [40,0,0,0]// 40 is a lower value of the price range
        Mode= [$ANY$,$Exact$,$Best$,$Best$]
        Essential= [$True$,$False$,$False$,$False$]
        Semantic= [$BetterLess$,$ANY$,$BetterMore$,$BetterMore$]

        // query-options preferences
        AllBest=$True$
end
```

4.3 The X-Algorithm Back-end

The back end of the algorithm is where the actual ranking performed. The X-Algorithm performs ranking in three different phases. These phases are: Preparation Phase, Multiplication Phase and Sorting Phase.

4.3.1 Preparation Phase

This phase is divided into two steps: the Measuring Step and the Scaling Step. In the Measuring Step, the attributes of *Query* and *Service* are involved in a pairwise comparison. This comparison produces unnormalized numbers that are based on different scales. Therefore, the results of the pairwise comparison proceeds to the next step which is Scaling Step. Thus, the goal of Preparation Phase is the following:

1. Perform pairwise comparison between the attributes of *Query* and available *Services*.
2. Consider the following parts of the user request: *Query*, Semantic, Mode, Range, and Algorithm Accuracy.
3. Normalize the results of pairwise comparison to a common scale.

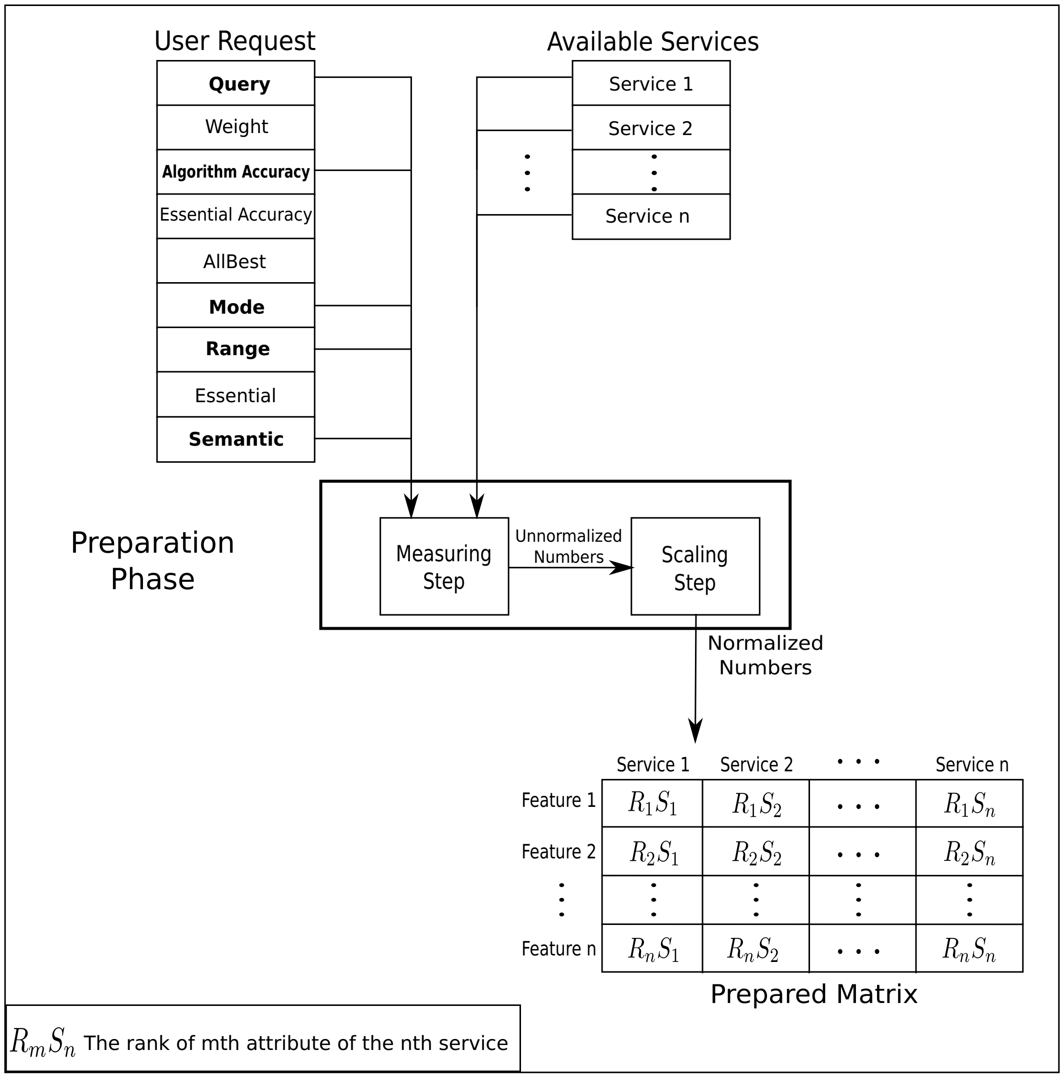


Figure 14: An illustration of the Preparation Phase

4. Generate the Prepared Matrix (PM).

In Measuring Step, the algorithm uses *Query*, Semantics, Mode, Range, and the Algorithm Accuracy as specified in the user request. In addition, the available list of *Services* is considered another input in this step. Hence, the algorithm takes each attribute of *Query* and compare it against an attribute of one *Service* using the defined *Similarity Measure* defined in next Chapter.

In Scaling Step, the produced numbers from the Measuring Step are justified into a common scale. Hence, each pairwise comparison result passes through a scaling method. This scaling method adjusts the results to one common scale that is used for all other results. Finally, results of the Scaling Step are recorded in the *Prepared Matrix*(PM). The rows of this matrix are the attributes of different *Services* with respect to the attributes of *Query*. That is, the i_{th} row is PM matrix contains the scores of the i_{th} attributes of available *Services* with respect to the i_{th} attribute in *Query*. The columns of PM matrix are the available *Services*. The PM matrix is generated using the following components from the user request: *Query*, Mode, Semantic, Range and Algorithm Accuracy. Also, the results in PM are all normalized to a common scale. Figure 14 illustrates both steps.

4.3.2 Multiplication Phase

In this phase, the results are tailored to the needs of consumers. In essence, the level of importance, whether essential or not, are used in this phase. Hence, Weights, Essential, and Essential Accuracy are considered in the calculations. This phase is introduced to perform the following tasks:

1. Calculate the total Weight vector (Regular Weight + Essential Weight)
2. Multiply PM by the Weight vector.
3. Generate the Unsorted Ranks (UR) list.
4. Apply AllBest preference based on the user request.

The level of importance introduced to the PM matrix by multiplying the Weight vector by the PM matrix. However, because Essential option is basically an extra weight assigned to the essential feature to outweigh the importance of other non-essential features, we divide Weight vectors to two sub-vectors; Regular Weight vector and Essential Weight vector. Hence,

$$Weight = RegularWeights + EssentialWeights$$

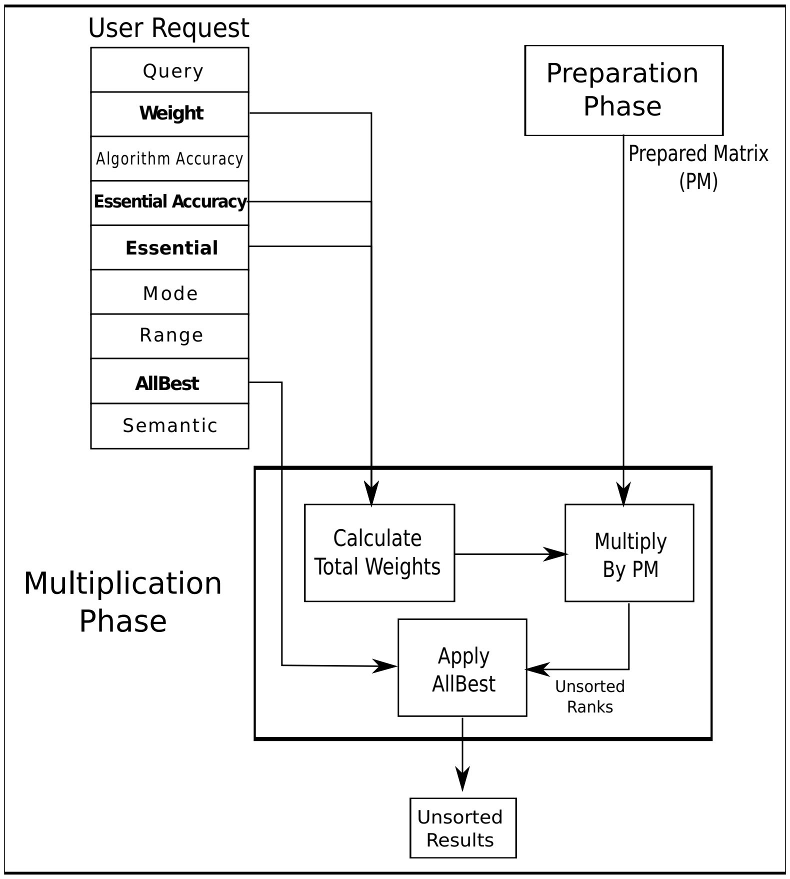


Figure 15: Illustration of the Multiplication Phase

After this addition, the Weight vector is multiplied by the PM matrix to produce what's called the *Unsorted Ranks* (UR). Hence,

$$UR = Weight * PM$$

$$UR = (weight_1, weight_2, \dots, weight_n) * \begin{pmatrix} S_{11} & S_{12} & \dots & S_{1n} \\ S_{21} & S_{22} & \dots & S_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ S_{m1} & S_{m2} & \dots & S_{mn} \end{pmatrix}$$

where,

S_{mn} is the score of the m_{th} feature of the n_{th} *Service*. Thus, the total rank of the i_{th} *Service* is calculated as,

$$TotalRank_{S_i} = weight_1 * S_{1i} + weight_2 * S_{2i} + \ldots + weight_n * S_{ni}$$

Since *UR* is a list of unsorted ranks of all *Services* ,

$$UR = [TotalRank_{S_1}, TotalRank_{S_2}, TotalRank_{S_n}]$$

UR is a list that contains the final ranks of all *Services* but these ranks are unsorted. This introduces us to the final phase, the Sorting Phase. In Sorting Phase there is one last user preference that is used which is AllBest option. Generally, AllBest preference prompts the algorithm to add values to the final rank as to prefer *Services* that meet at least all the minimum requirements of the *Query* . The added value is obtained based on some calculations introduced in Chapter 5. Figure 15 depicts the Multiplication Phase.

4.3.3 Sorting Phase

This phase receives UR vector from the Multiplication Phase and applies a sorting algorithm to sort the list. Any sorting algorithm can be employed to sort the list in non-increasing order of ranks. However, we preferred the Quick Sort algorithm because it is stable [CLRS09]. The output of this phase is what we call Sorted Ranks (SR). Thus, this phase is to perform the following tasks:

1. Apply a sorting algorithm on UR vector and sort it in non-increasing order.

2. Generate the Sorted Ranks (SR).

4.3.4 General Pseudo Code of the Back End

In this subsection, we provide an abstract pseudo code of the back end. This pseudo code is shown in Algorithm 2.

Algorithm 2: General Pseudo Code of the X-Algorithm Back End

```
begin
    // Preparation Phase
    Input: query, range, semantic, algoAccuracy, mode, ServicesList
    Output: PM matrix
    for $i \leftarrow NumberOfServices$ do
        $service \leftarrow ServiceList[i]$
        for $j \leftarrow NumberOfFeaturesInQuery$ do
            $preferences \leftarrow [range, mode, algoAccuracy, semantic]$
            $unnormalizedRank \leftarrow$
            $SimilarityMeasure(service[j], query[j], preferences)$
            $normalizedRank \leftarrow ScalingMethod(unnormalizedRank)$
            $PM_{ij} \leftarrow normalizedRank$
        end for
    end for

    // Multiplication Phase
    Input: essential, essentialAccuracy, AllBest, weights, PM
    Output: UR list
    $essentialWeight \leftarrow calculateEss(essential, essentialAccuracy)$
    $TotalWeight \leftarrow Weights + essentialWeight$
    $UR \leftarrow DotProduct(TotalWeight, PM)$
    $UR \leftarrow ApplyAllBest(AllBest, UR)$

    // Sorting Phase
    Input: UR list
    Output: SR list(Descending Order)
    $SR \leftarrow QuickSortDescending(UR)$

end
```

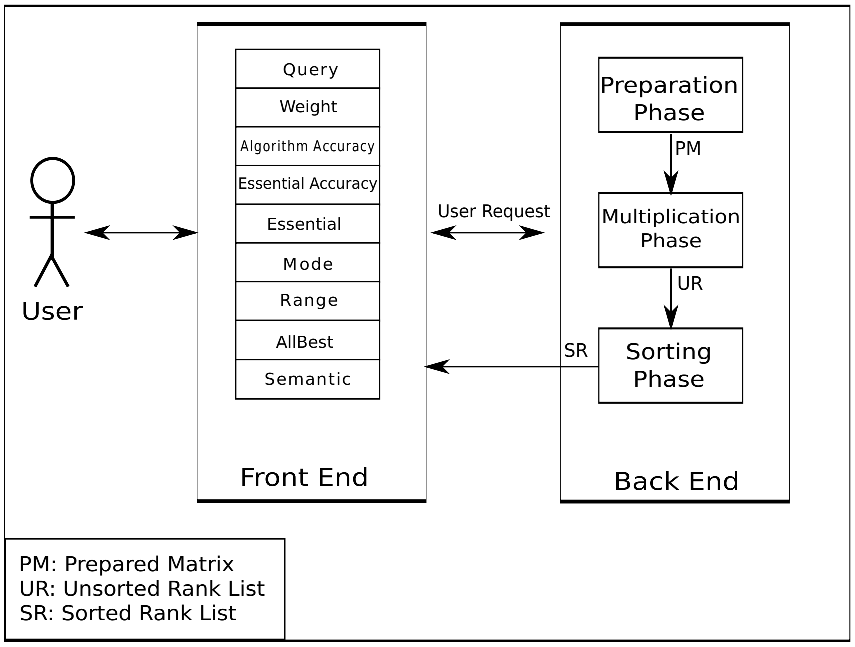


Figure 16: Illustration of the entire structure of X-Algorithm

4.4 Summary

In this chapter we have introduced the basic concepts of the X-Algorithm. We explained the X-Algorithms from two different views: front end back end. For each view, we discussed its functionality and its pseudo code. Figure 16 summarizes the structure of the X-Algorithm.

Chapter 5

Deeper Look Into The X-Algorithm

In this chapter, we expand the general information provided in previous chapter with details. We discuss each feature of the X-algorithm in terms of why and how it was integrated. Also, we support each option by a comprehensive example that touches all dimensions of the option. Finally, we provide an inclusive pseudo code that takes into consideration all options and characteristics of the algorithm.

5.1 Weights

Weight is a non-negative number assigned to represent level of importance for each feature in a *Query*. Thus, Weight Vector has the same length as *Query*. This vector is to be eventually multiplied by a matrix to be defined later in order to produce the ranks of *Services*. The different kinds of weights are defined as follows:

1. Significant ($weight = 0.005$): This weight is the highest weight (w_h) in our algorithm. It is assigned to a feature with highest significance.

2. Normal ($weight = 0.003$): This weight is assigned to a feature of a normal importance. Most of the features are usually at this level of importance.

3. Low ($weight = 0.001$): This weight is assigned to a feature if it is not considered important, yet its existence causes measurable impact on the generated ranks.

4. Insignificant($weight = 0.0001$): The feature assigned this weight is not considered by the algorithm unless it is required to discriminate between equal *Service* rankings. That is, if two *Services* have all features of similar quality and differ from each other only because of an insignificant feature, the *Service* with better insignificant feature ranks higher. This means that there is a slight consideration given for insignificant feature. This weight (w_l) is the lowest among the weights .

5. Do Not Consider($weight = 0$): A feature of this weight will not be considered in the generated ranks even if qualities of other features match. Users can achieve the same results by not including this feature in *Query* definition at all.

The defined weights have two main characteristics. First, they are small. The reason is that in our ranking algorithm, we construct a matrix by integrating many options that may increase the matrix elements. If the weights are not small numbers, when we multiply this matrix with the weight vector the final rank value may grow exponentially and become unmanageable. Second, the difference between weights is set as 0.002. In previous works [Ibr12] sequential weights, such as $1, 2, 3, \ldots$ or $0.01, 0.02, 0.03, \ldots$ have been used. Based on our practical observations we found that sequential weights of small order, such as 0.001 and 0.002, are not very effective and can easily make almost similar results. Based on empirical study, we found 0.002 to be sufficient to make the desirable difference. So, we use a distance of 0.002 between weights to generate measurable difference between the ranks.

5.2 Modes, Semantics and the value of X

We introduce modes as a means to provide more options to users in specifying their preferences and to simulate real life experience. In this section, we introduce the different ranking modes offered by the X-Algorithm. Also, we discuss how the modes give the basis to introduce semantic-awareness to the X-Algorithm. Then, we introduce the semantics of data encountered by the algorithm. After that, we integrate the semantics into our algorithm. We follow this by a pseudo code illustrating how all semantics are handled in the X-Algorithm. Finally, we provide an example illustrating that our method is able to rank based on the semantic of inputs.

5.2.1 Ranking Modes

When we look for *Services* in real life, we have two types of requirements. First type of requirement is met by exact matching. Examples of such requirements include finding the number of rooms in a hotel, the model of a car that can accommodate five passengers, and products with specific brand names. Second type of requirement is met through the minimum needs. Examples of this type include booking a hotel which has at least 3 stars, finding a product with at least 1 year warranty, and finding a job that pays at least \$50,000 per year. In some online *Services* we want to be able to look for exact values to some features of the *Query*, and better values for other features. As an example, in online film renting *Service* we usually want to see a specific film while we always look for better/cheaper price. Thus, we want the name feature of the film *Service* to be an exact match and the price/cost feature to be a better value. Therefore, we introduce two modes of ranking in our algorithm, called *Exact Mode* and *Best Mode*. The *Exact Mode* prompts the algorithm to look for an exact match to a specific feature in a *Query*, while *Best Mode* prompts it to look for better values than a specific feature in *Query*.

However, before diving into the details of *Exact Mode* and *Best Mode*, we have to define the meaning of the word "better". In some cases, “Less is Better” and in some others “More is Better”. This motivates us to study three kinds of semantics of the data encountered by the X-Algorithm. These are (1) Less is Better(LB), (2) More is Better(MB), and (3) Exact is Better(EB).

5.2.2 Semantics

Generally, data that we use to seek online *Services* include numerical and non-numerical types. Examples of numerical data include cost, weight, and percentage. Examples of non-numerical data include product names, product codes, and addresses. Numerical data can be data that are better as they increase in value, such as quality-related data. Examples are star ranking of a hotel, recommended rating of a *Service* , and reliability. It is also possible that some numerical data are considered better as they decrease in value. Examples include cost, waiting time in between repairs, and shipping time. The kinds of non-numerical data that we come across are of string or boolean types. String types, for names and addresses, must have an exact match. Boolean values, such as availability of a property in a product, which is described as "True/False" must be matched exactly. We summarize below our discussion on the semantic classification of data encountered by the X-Algorithm. Figure 17 illustrates the summary.

1. Numerical: There are two types.
    - better as the value increase (MB), and
    - better as the value decrease (LB).
2. Non-numerical: There are two types.
    - String values (EB), and
    - Boolean values (EB).

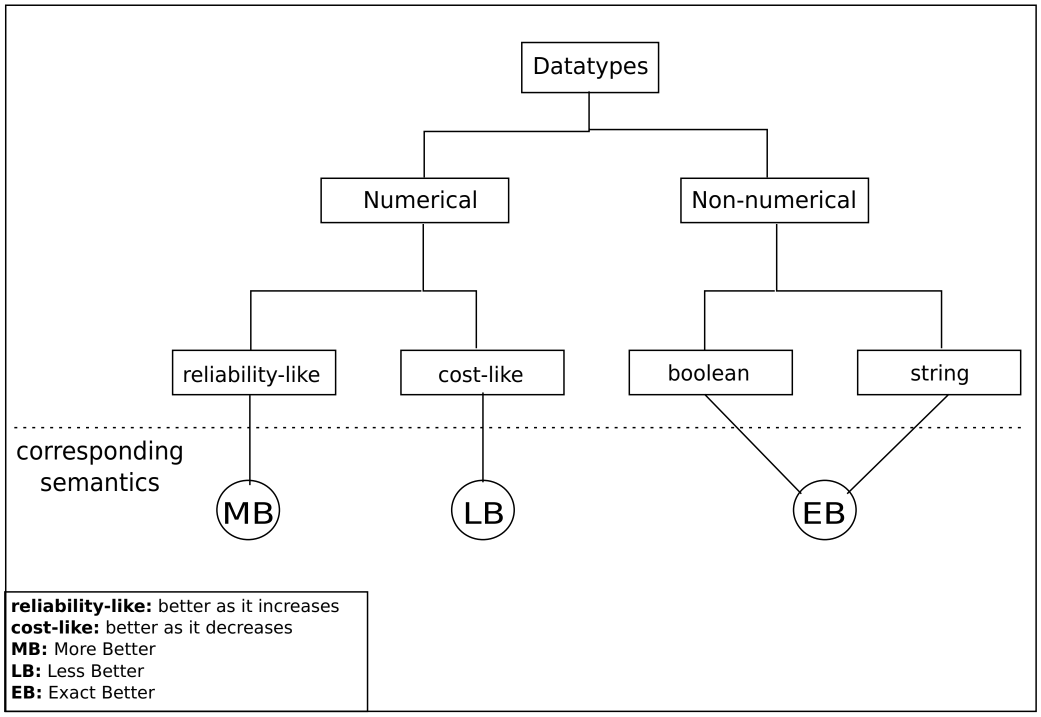


Figure 17: Classification of semantics based on data types

5.2.3 Defining Scores Types

In this section, we define two types of scores, called Reward Score ($reward$) and Penalty Score ($penalty$). These two types of scores will be generated by our *Similarity Measure*. In later sections we discuss how to assign semantics in different modes.

1. Reward Score($reward$): This score is assigned to the attribute of *Service* that offers an equal value or better one than the one defined in *Query*. It can be a single value that is assigned in one specific case, or a range of values that fluctuates between a minimum reward ($minReward$) and a maximum reward ($maxReward$). The value of $minReward$ is fixed for all modes and semantics. It is also the value assigned when Reward Score is a single value, while the value of $maxReward$ might be different for different semantics and modes, as will be explained in later sections.

2. Penalty Score($penalty$): This score is assigned when the attribute of a *Service* has a value

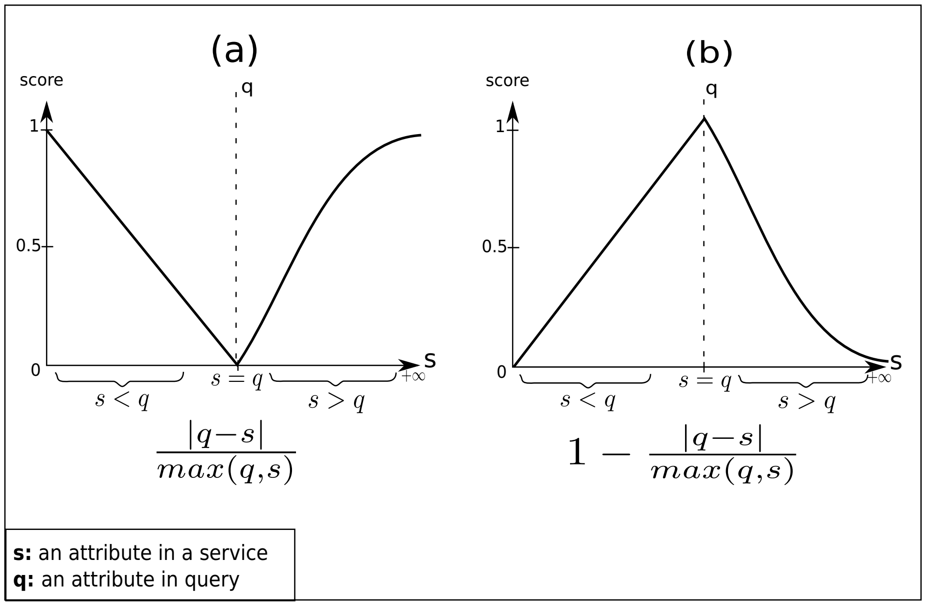


Figure 18: The difference in RC behaviour when subtracted from "1"

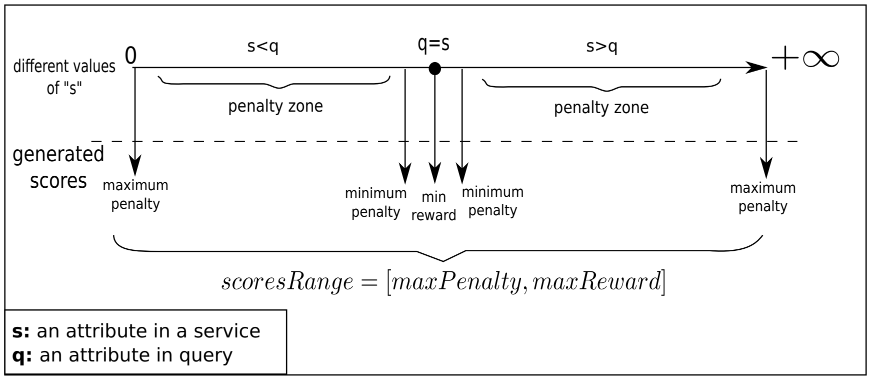


Figure 19: Different scores assignment in *Exact Mode*.

that is worse than the value required in *Query*. Unlike $reward$, $penalty$ is always in the range $[minPenalty$ to $maxPenalty]$. The value of $minPenalty$ is a value less than the value of $minReward$. The $penalty$ score will approach the value of $maxPenalty$ as the provided value in *Service* gets worse than the value defined in user *Query*.

Having introduced modes and score types we explain in the following sections how scores are assigned at different modes. We recall the definition of Relative Change RC (Q) *Similarity Measure* introduced in Chapter 3.

5.2.4 Exact Mode

By definition, RC is to generate scores that represent the similarity between two numerical inputs. The RC *Similarity Measure* generates an output in the range $[0, 1]$. Let q and s be numerical values, where q is related to an attribute in *Query*, and s is related to the same attribute in a *Service*. RC *Similarity Measure* considers the three different cases $q < s$, $q > s$ and $q = s$. Basically, RC generates the lowest score when $q = s$, and it generates graded score that approaches 1 as the difference between q and s increases (for cases $q < s$ and $q > s$). However, in *Exact Mode* we want the opposite behaviour. Specifically, we want the $reward$ value to be assigned when exact match occurs ($q = s$), and we want a decreasing graded score that approaches 0, namely increasing $penalty$ value, as the difference between values of q and s increases. Because the scores are bounded above by 1, the behaviour of RC can be reversed by subtracting the generated scores from 1. Thus, the highest score will be generated when $q = s$ and graded scores approaching 0 will be generated when the deviation between s and q gets higher. We used this reversing function $1 - RC$, whose behaviour is shown in Figure 18, to be able to generate proper scores. Figure 19 shows how the values of $penalty$ and $reward$ are defined for *Exact Mode*. Based on this discussion we define,

$$RC_{EXACT} = \begin{cases} 1 & s = q \\ 1 - \frac{|q-s|}{max(q,s)} & s < q, s > q \end{cases} \tag{3}$$

Remark. *The benefit of normalization is brought out in our approach. Because the results of RC are bounded above by* 1 *and the calculations are normalized to lie in the interval* $[0, 1]$*, we are able to reverse the behaviour of the RC results.*

5.2.5 Best Mode

For *Best Mode*, we need to understand what "better" means for each semantic. Then, we try to incorporate this meaning into our *Similarity Measure*. However, the problem associated with

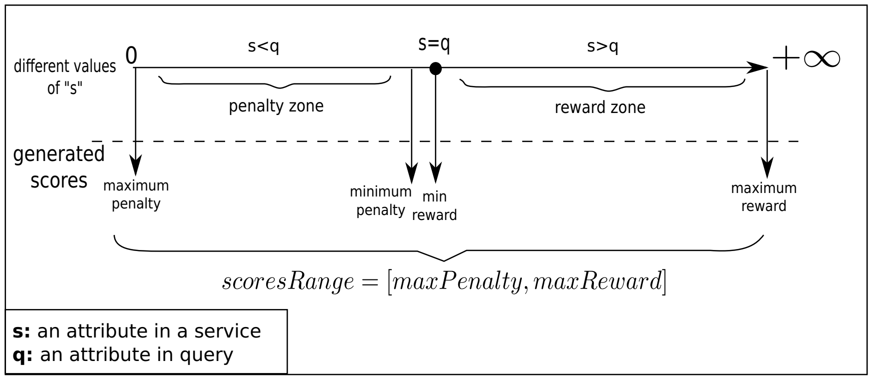


Figure 20: Types of Score assignment for More is Better Semantic

finding better values is that we don't know what is the best value in the available *Services*. Unlike *Exact Mode* where we assign $reward$ to exact match, when finding better values there is no specific value to which we can assign $rewards$. To overcome this obstacle, we assign $reward$ and $penalty$ differently for the three types of semantics MB, LB, and EB.

MB Semantic

In MB semantics "More is Better". Fixing on the same attribute in a *Service* and *Query*, this means that the *Service* in which this attribute value is higher than the attribute value in the user *Query* is to be preferred. That is, a higher value in *Service* is better than a lower one. Thus, the value defined in the *Query* is the minimum acceptable value for the client. For example, when we look for a *Service* that rates 3/5 or higher, we are using MB as a semantic for the rating feature. In technical terms, we want our *Similarity Measure* to assign higher $reward$ for features with higher values. Hence, $minReward$ should be assigned when once minimum requirement is met, and the $reward$ value should be increased towards $maxReward$ as the value of *Service* attribute goes higher than the value defined in *Query* . On the other hand, the $penalty$ value should be applied when the value of *Service* attribute is less than the value of the *Query* attribute and it should approach $maxPenalty$ as the value of *Service* attribute goes further below the value defined in the *Query*. Assume s denotes the value of a *Service* attribute, and q denotes the value of the same attribute in $Query$. We define the RC function for MB as shown below:

$$RC_{MB} = \begin{cases} penalty & s < q \\ minReward & s = q \\ v, v \in [minReward, maxReward] & s > q \end{cases}$$

Figure 20 shows how the score types are defined for MB semantic. We need to assign values to $penalty$, $minReward$, and v in such a way that $penalty < minReward < v < maxReward$, and v approaches $maxReward$ as $s - q$ increases. We fix $minReward$ at 1, and adapt RC function from Figure 18 (b) to define $penalty$ and v. We find that the left half of Figure 18 (b) meets our requirement. That is, when s and q values are close the score is high. However, when s exceeds the value of q the score drops down towards 0 while we want it to continue increasing. To motivate what we do to achieve this behaviour, we take a simple example. Assume $q = 10$, and we have different values of s that range in the interval $[0, 20]$. Using the formula $1 - \frac{|q-s|}{max(q,s)}$ from Figure 18 we obtain Table 6.

q	s	case	RD
10	0	$s < q$	$1 - 1 = 0$
10	2	$s < q$	$1 - 0.8 = 0.2$
10	6	$s < q$	$1 - 0.4 = 0.6$
10	8	$s < q$	$1 - 0.2 = 0.8$
10	10	$s = q$	$1 - 0 = 1$
10	12	$s > q$	$1 - 0.167 = 0.833$
10	14	$s > q$	$1 - 0.286 = 0.714$
10	16	$s > q$	$1 - 0.375 = 0.625$
10	18	$s > q$	$1 - 0.444 = 0.556$
10	20	$s > q$	$1 - 0.5 = 0.5$

Table 6: Results of $1 - RC$ applied on $q = 10$, and $s = [0, 20]$

In this table, we find that the computed results for $s \leq q$ are consistent with MB requirement. On the other hand, scores generated for values of $s > q$ show a behaviour that is quite opposite to what we want. This seems to be because of the 1 that we subtract the score from. Therefore, we remove 1 from the case $s > q$ as an attempt to keep the increasing behaviour of the scores. Now we obtain scores as shown in Table 7.

q	s	case	RD
10	0	$s < q$	$1 - 1 = 0$
10	2	$s < q$	$1 - 0.8 = 0.2$
10	6	$s < q$	$1 - 0.4 = 0.6$
10	8	$s < q$	$1 - 0.2 = 0.8$
10	10	$s = q$	$1 - 0 = 1$
10	12	$s > q$	0.167
10	14	$s > q$	0.286
10	16	$s > q$	0.375
10	18	$s > q$	0.444
10	20	$s > q$	0.5

Table 7: Results after removing 1 from the case $s > q$

Results in Table 7 shows an increasing behaviour for the scores generated for case $s > q$. Yet, because the results of the *RC Similarity Measure* are bounded above by 1, it does not exceed the value of 1. Therefore, scores need to be “scaled”, which we achieve by manually shifting the scores by 1. Table 8 shows the result after shifting the scores of case $s > q$ by 1. This shifting will change the original upper bound of *RC* from 1 to 2. Thus, MB semantics will produce bounded scores.

q	s	case	RD
10	0	$s < q$	$1 - 1 = 0$
10	2	$s < q$	$1 - 0.8 = 0.2$
10	6	$s < q$	$1 - 0.4 = 0.6$
10	8	$s < q$	$1 - 0.2 = 0.8$
10	10	$s = q$	$1 - 0 = 1$
10	12	$s > q$	$0.167 + 1 = 1.167$
10	14	$s > q$	$0.286 + 1 = 1.286$
10	16	$s > q$	$0.375 + 1 = 1.375$
10	18	$s > q$	$0.444 + 1 = 1.444$
10	20	$s > q$	$0.5 + 1 = 1.5$

Table 8: Results after adding 1 to the case $s > q$

The results in Table 8 shows the behaviour we need under MB semantics. We come to the

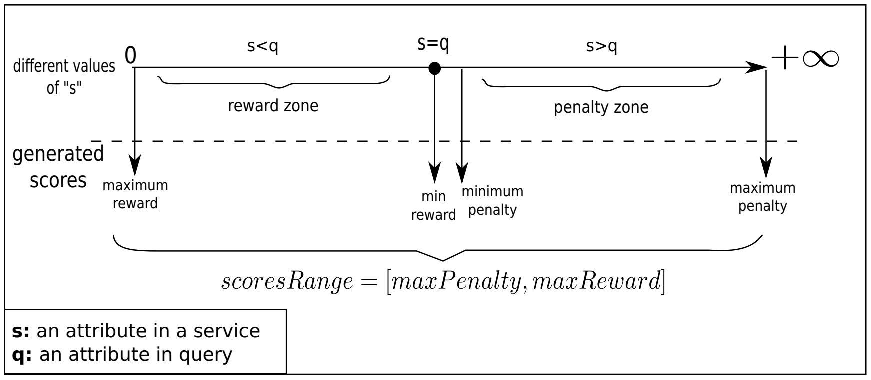


Figure 21: Types of scores assignment for Less is Better Semantic

following definition of score:

$$RC_{MB} = \begin{cases} |-1| & s = q \\ |-1 + \frac{|q-s|}{q}| & s < q \\ |-1 - \frac{|q-s|}{s}| & s > q \end{cases} \tag{4}$$

Remark. *Now the range of scores generated under MB semantics is different from the range for scores generated in EM semantics. This shows that scaling method is necessary when we fix RC function.*

In Equation 4, the operation performed when the case is $s < q$ is subtraction, where the operation performed when the case is $s > q$ is addition. Changing the sign of -1 would result in reversing the operations performed and thus reverse the overall outcome. Also, we used $|-1|$, instead of 1, just for the sake of uniformity and explicitly pointing out the scaling.

LB Semantic

The Semantic LB means "Less is Better". It indicates that the less the value for a specific feature in a *Service* the better it is. Figure 21 shows how different score types are assigned for *LB* semantic. For example, price is usually better when it is less, i.e. cheaper. Thus, the behaviour we expect with this semantic is the opposite to the one defined for MB. Therefore, it is sufficient to changing -1 in Equation 4 to $+1$ to arrive at the formula for LB semantics.

$$RC_{LB} = \begin{cases} |+1| & s = q \\ |+1 + \frac{|q-s|}{q}| & s < q \\ |+1 - \frac{|q-s|}{s}| & s > q \end{cases} \tag{5}$$

In Equation 5, as opposed to MB, the operation performed in the case $s < q$ is addition, i.e. a reward is granted, where the operation performed in the case $s > q$ is subtraction, i.e. a penalty is assigned. Now, to combine both semantics in one definition, we refer to the semantic by X. We define the value of X as follows,

$$X = \begin{cases} -1 & \textit{Best Mode}, semantic = MB \\ +1 & \textit{Best Mode}, semantic = LB \end{cases} \tag{6}$$

Using the definition of X from Equation 6 in Equation 4 and Equation 5, we arrive at the definition of RC_X *Similarity Measure*. This definition is *semantic-aware* and is used in *Best Mode*.

$$RC_{X} = \begin{cases} |X| & s = q \\ |X + \frac{|q-s|}{q}| & s < q \\ |X - \frac{|q-s|}{s}| & s > q \end{cases} \tag{7}$$

EB Semantic

The Semantic *EB* means "Exact is Better". In this semantic exact match is possible, because *EB* includes string and boolean data types in which the concept “better values” does not exist. *EB* semantic is based on "*match or mismatch*" concept where there is no need to associate modes with *Service* requests. So, we can decide to assign $reward$ when equality occurs between an attribute value in a *Service* and the value of the same attribute in *Query*, and assign a penalty when mismatch occurs. However, we wish to add to the algorithm more flexibility to users and simulate real life experience. So we decided to allow a *Query* attribute for *EB* to be defined as a set of options (string values). For example, if a user is looking for a laptop where its colour can be either black or white, the attribute value for colour attribute in the *Query* can be specified as $colour = \{black, white\}$. In turn, the algorithm would grant the $reward$ to the laptop whose colour is included in the set of options provided by the user. Since each attribute in a *Service* has a unique value s, whereas the user can specify set of values q for the same attribute in a *Query* we use the set notation ($s \in q$) to denote the presence of *Service* attribute s in the set of values q defined in *Query*. We write ($s \notin q$) to denote the absence of s in q. Even with this slight extension we are still requiring a full match.

We allow another extension in which partial match between *Service* feature and user specified feature is possible. This partial match is restricted to string values. For example, if q is defined as a set of strings $'x'$ and $'y'$ such that $q = ['x','y']$, and the attribute of a *Service* was found to be $s =' xm'$, the algorithm won't consider the partial match between x(the value of first option of q) and $'xm'$(the value of s). Thus, if we consider the concept of "*match/mismatch*" only, ranking *EB* type becomes very limited. To overcome this limitation we consider the possibilities that q_i is a subset of s or s is a subset of q_i, as an attempt to introduce partial match. We denote these situations by ($q \subset s$) and ($s \subset q$), respectively.

The partial match, ($q \subset s$) or ($s \subset q$), should receive a $reward$ than the mismatch, ($s \neq q$) and lower score than the exact match ($s = q$) or ($s \in q$). For example, consider that we are looking for a product under the names 'dictionary' or 'oxford dictionary', and the *Query* attribute that represents the name of dictionary is q, where,

$$q = ['dictionary','oxford\_dictionary']$$

Suppose the attributes s_1, s_2, and s_3 are the name attributes of three different available *Services* where,$s_1 =' dictionary'$, $s_2 =' oxford'$ and $s_3 =' translator'$. Based on our definition, s_1 should get the $reward$ (full match), s_2 gets a lower rank (partial match) and s_3 gets the lowest rank (mismatch).

In Chapter 3, we introduced Feature-Based *Similarity Measures* that examine only equality between features of *Query* and *Service*. In the aforementioned discussion, we have extended the Feature-Based *Similarity Measure* to include partial matches. The symbol SM_{EB} denotes the Feature-Based *Similarity Measure* used with *EB* semantic. In order to be consistent with the highest and lowest ranks produced for *MB* and *LB* semantics, within bounds, we define SM_{EB} as below:

$$SM_{EB} = \begin{cases} +1 & s = q \text{ , } s \in q \\ 0.5 & s \subset q, q \subset s \\ 0 & s \nsubseteq q \end{cases} \tag{8}$$

Remark. *Boolean Values are based on Match/Mismatch concept. The above discussion is irrelevant to boolean since it only has two values (True/False).*

To arrive at a uniform notation, we need to adjust the values of X to include the *EB* semantics. Since *EB* is not involved in RC, X can be any value different from the values used for *MB* and *LB*. We choose this value to be $Zero$. Hence,

$$X = \begin{cases} 0 & EB \\ +1 & LB \\ -1 & MB \end{cases} \tag{9}$$

Once the algorithm detects $X = 0$, it calculates the output based on Equation 8. Now, we have integrated all semantics in *Best Mode*. Also, we have integrated *Exact Mode* earlier. We want to make a unified definition for numerical types and another one for non-numerical types. This is to simplify implementation process. This requires us to include *Exact Mode* in the definition of X.

$$X = \begin{cases} 0 & EB \\ +1 & LB \\ -1 & MB \\ +1 & Exact\ Mode, s > q \\ -1 & Exact\ Mode, s < q \end{cases} \tag{10}$$

Finally, the definition of similarity measure is

$$SM_X = \begin{cases} SM_{EB} & X = 0 \\ RC_X & (Best\ Mode\ or\ Exact\ Mode)\ and\ (X = -1\ or\ X = 1) \end{cases} \tag{11}$$

where RC_X is defined in Equation 6 and SM_{EB} is defined in Equation 8

5.2.6 Similarity Measure Pseudo Code

Hereunder is the pseudo code for SM_X, which includes all situations of X.

Algorithm 3: Psuedo code for SM_X for handling different data types (MB,LB and EB) and for modes Exact and Best based on the value of X

```
Result: Services ranked based on value of X
Input: $i_{th}$attribute of Query(q),$i_{th}$attribute of Service(s),X
begin
    if $semantic == EB$ then
        $X \leftarrow 0$
    else
        if $Mode == Exact$ then
            if $s < q$ then
                $X \leftarrow -1$
            else
                $X \leftarrow +1$
        else    // $Mode = Best$
            if $semantic == LB$ then
                $X \leftarrow +1$
            else
                $X \leftarrow -1$
    if $X$==$0$ then    // EB semantic
        if $s = q$ $OR$ $s \in q$ then
            $R \leftarrow 1$
        else if $q \subset s$ $OR$ $s \subset q$ then
            $R \leftarrow 0.5$
        else
            $R \leftarrow 0$
    else    // MB,LB semantics for both modes (Exact&Best)
        if $s = q$ then
            $R \leftarrow |X|$
        else if $s < q$ then
            $R \leftarrow |X + \frac{(q-s)}{q}|$
        else
            $R \leftarrow |X + \frac{(s-q)}{s}|$
end
```

Example 5. *This example illustrates the outcome of Algorithm 3 when it is applied to a Query which has different data types and both ranking modes. The user request Q and four Services S_1, S_2, S_3, and S_4 are defined in Table 9 and Table 10 respectively.*

Features' names	Price(f_1)	Rating(f_2)	Colour(f_3)	Weight(f_4)
Features' Semantics	LB	MB	EB	LB
Ranking Mode	Best	Best	Best	Exact
User Query	1000 CAD	4 stars	['RED',BLACK']	4 Pounds

Table 9: User Request

Services	Price(f_1)	Rating(f_2)	Colour(f_3)	Weight(f_3)
S_1	900 CAD	9 stars	'RED'	6 Pounds
S_2	1500 CAD	7 stars	'WHITE'	2 Pounds
S_3	400 CAD	3 stars	'BLACK'	3 Pounds
S_4	1000 CAD	4 stars	'BLACK&WHITE'	2.7 Pounds

Table 10: The values of features of each Service

According to Algorithm 3, we calculate the scores for each feature and their total. The result is shown in Table 11.

Services / Ranks	Price(f_1)	Rating(f_2)	Colour(f_3)	Weight(f_4)	Total Rank
S_1	1.1	1.56	1	0.67	4.33
S_2	0.67	1.43	0	0.5	2.6
S_3	1.6	0.75	1	0.75	4.1
S_4	1	1	0.5	0.675	3.175

Table 11: The ranks of each Service based on the X-Algorithm

We notice that although values of $S_{1_{f_4}}$ and $S_{2_{f_4}}$ have the same distance from Q_{f_3}(the third column in Table 9) they have different ranks. However, based on *Exact Mode* definition the rank should be produced based on the distance from the *Query* and thus they should have the same ranks. So, Algorithm 3 needs to be sharpened. This issue is resolved in next section.

5.3 Symmetry in Exact Mode and Semantic Preference

In this section we discuss symmetry and semantic preference (SP). In Chapter 3 we have defined symmetry property of *Similarity Measure*. We use the same notation as before. Fixing the attribute, we let q denote its value in a *Query*, and s_1 and s_2 denote its values in two *Services*. *Similarity Measure* has similarity property if it produces the same rank for the attribute in the two *Services* if $\mid q - s_1 \mid = \mid q - s_2 \mid$. We say that values s_1 and s_2 are 'images' around q. Clearly, one of the *Service* value is higher than the other. In *Exact Mode*, the user might prefer the lower value (as in cost) if no exact match is possible. So, we discuss Semantic Preference(SP) to assign higher rank to *Service* attribute that is preferred by the user. It is clear that SP is dependant on the presence of symmetry.

5.3.1 Symmetry

Suppose a consumer wants to rent an apartment for \$1000/$month$. However, there is no apartment matching the user requirement. Assume the closest available options are apartments with rents \$800/$month$ and \$1200/$month$. Since both options differ by the same difference from the defined requirement, both should capture a similar degree of the consumer's interest. That is, the *Similarity Measure* should produce similar scores for both options.

Our *Similarity Measure* RC_X for *Exact Mode*, as defined in Equation 3 in Section 5.2, looks like,

$$RC_{EXACT} = \begin{cases} |1| & s = q \\ |-1 + \frac{|q-s|}{q}| & s < q \\ |+1 - \frac{|q-s|}{s}| & s > q \end{cases} \tag{12}$$

RC_{EXACT} does not have symmetry property. It is due to the different denominators in the fractional parts of its definition for the cases $s > q$ and $s < q$. In order to achieve symmetry we have to unify the denominators, without violating the bounded outputs and normalization properties. We first tried the fraction

$$Fraction_{asymmetric} = \frac{\triangle}{max(q,s)}$$

where $\triangle = |q-s|$ to replace the fractions $\frac{|q-s|}{q}$, and $\frac{|q-s|}{s}$. This does not produce the desired results, as illustrated in Table 12 for $q = 10$, $s_1 = [0,2,4,6,8]$, and $s_2 = [20,18,16,14,12,10]$. Since $|s_2 - q| = |s_1 - q|$, the *Service* attribute values are symmetric with respect to the *Query* attribute value, however their ranks are not equal.

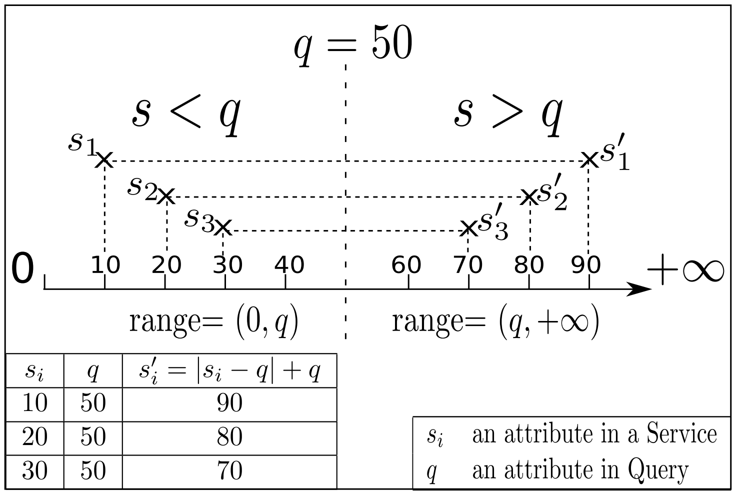
$Fraction_{asymmetric}$ $(q = 10)$				
$\|q-s_1\| = \|q-s_2\|$	$s_1 \uparrow$	$s_1 < q$	$s_2 \downarrow$	$s_2 > q$
10	0	1	20	0.5
8	2	0.8	18	0.44
6	4	0.6	16	0.38
4	6	0.4	14	0.29
2	8	0.2	12	0.17

Table 12: The asymmetric results produced by $Fraction_{asymmetric}$

To remedy the situation, we need to understand the behaviour of the function $\frac{\triangle}{max(q,s)}$, for $s \in (0,\infty)$ and for a fixed q. Suppose $s_1 > q$ and $s_2 < q$. For the case $s_1 > q$ the denominator $max(q,s_1) = s_1$ of $Fraction_{asymmetric}$ is in the range (q,∞). For the case $s_2 < q$ the denominator $max(q,s_2) = q$ of $Fraction_{asymmetric}$ is not in the range (q,∞), because it is an open interval. We need to push this denominator by the amount $s_2 - q$, which is the image of s_2 around q. This will give the new denominator $s_2' = |s_2 - q| + q$, which lies in (q,∞) to the case $s_2 < q$. In the light of this discussion, we define $s' = \triangle + q$, where $\triangle = |q-s|$, as the denominator for case $s < q$, and $s = max(q,s)$ as the denominator for the case $s > q$. Figure 22 illustrates the achieved symmetry for a simple example.

Thus we arrive at the refined definition $Fraction_{symmetric}$:

$$Fraction_{symmetric} = \begin{cases} \frac{\triangle}{s'} & s < q \\ \frac{\triangle}{s} & s > q \end{cases} \tag{13}$$



s_i	q	$s'_i = \lvert s_i - q\rvert + q$
10	50	90
20	50	80
30	50	70

Figure 22: Example of the method used to unify denominators

where, $\triangle = |q - s|$, $s' = \triangle + q$

By applying the fraction $Fraction_{symmetric}$ on the same example given above we obtain the results listed in Table 13. Now, the symmetry property is restored.

$Fraction_{symmetric}$ $(q = 10)$					
$\lvert q - s_1\rvert = \lvert q - s_2\rvert$	s'_1	$s_1 \uparrow$	$s_1 < q$	$s_2 \downarrow$	$s_2 > q$
10	$\lvert 10 - 0\rvert + 10 = 20$	0	0.5	20	0.5
8	$\lvert 10 - 2\rvert + 10 = 18$	2	0.44	18	0.44
6	$\lvert 10 - 4\rvert + 10 = 16$	4	0.38	16	0.38
4	$\lvert 10 - 6\rvert + 10 = 14$	6	0.29	14	0.29
2	$\lvert 10 - 8\rvert + 10 = 12$	8	0.17	12	0.17

Table 13: The symmetric results produced by $Fraction_{symmetric}$

Remark. *When we unified the denominators, we have caused the bounds of the output to fluctuate. We might need to scale the results for reasons that are already highlighted in the previous section. Scaling method is discussed in next section.*

5.3.2 Semantic Preference

Semantic Preference(SP) is the process of scoring an attribute of a *Service* higher than an attribute of another *Service* because it is semantically better. For example, assume the price of two *Services* has the same difference from the price defined in *Query* such that the price of the first *Service* is more than *Query* and the price of the second *Service* is less than *Query*. Consumers tend to prefer the second *Service* as it is cheaper than the first *Service*. This is the meaning of "semantically better".

Since scores produced for symmetric attributes of different *Services* with respect to *Query* are identical, we can introduce a manual change on the *Similarity Measure* to enforce the semantic preference. Basically, this manual change is a value that is added to the *Similarity Measure* of the semantically better attribute which we call the $BoostValue$. However, since we have many available *Services*, when we perform the manual change, we have to be aware not to disturb the fairness of the scores. This implies the following points:

- The introduction of *BoostValue* should not affect other characteristics of the *Similarity Measure*, such as normalization, sensitivity to small changes, and simplicity.
- The introduction of *BoostValue* should allow semantic preference within the symmetric attributes and not adversely affect any other attribute.

To clarify, let $score_1$,$score_2$ and $score_3$ be the scores of the price feature in three different *Services*. Suppose their values are $1.2, 1.2, 1.3$, and $score_1$ and $score_2$ are scores of symmetric attributes. Now, the *BoostValue* that does not disturb the fairness should not make 1.2 better than 1.3, and should only affect the scores of symmetric attributes. This is challenging because we don't know how close the other scores are to the scores of symmetric attributes. In other words, we don't know how small should be the value of *BoostValue* that will not affect other surrounding scores related to other attributes. In the case of our example, *BoostValue* can be any value that is smaller than 0.1. But what if there is another score, say $score_4 = 1.21$, then *BoostValue* should be less than 0.01. Therefore, there has to be an acceptable error rate specified for each application domain. In order to achieve these goals, we make a change in the *Similarity Measure* definition (of symmetry) rather than change the scores computed by it. We need to refine the symmetric *Similarity Measure* for *Exact Mode* which has been defined as follows:

$$RC_{Exact} = \begin{cases} |-1+\frac{\triangle}{s'}| & s < q \\ |+1-\frac{\triangle}{s}| & s > q \end{cases} \tag{14}$$

where, $\triangle = |q-s|$,$s' = \triangle + q$, q an attribute in *Query*, s an attribute in *Service*, and s' an image of s around q.

In order to achieve the first requirement of *BoostValue*, we introduce it to change the denominator of RC_{Exact}. This will ensure that the value of the fraction won't exceed 1 and thus the total won't exceed the maximum bound. Also, because the number to be added is small, it will not affect the sensitivity of the *Similarity Measure*. Moreover, because it is just a numerical value it will not affect the simplicity of the *Similarity Measure*. However, to make sure it won't affect other attributes related to other *Services*, we have to define an acceptable accuracy that is specific to consumers/application domain. However, the question is what is the $Boost\ Value(BV)$ that can be added to the denominator and shift all the semantically better features within a specific accuracy level? A formal justification of finding this value is in order.

Assume we have a feature f_a and f_b which are attributes in $service_a$ and $service_b$ respectively. Assume that both f_a and f_b have the same difference from the value of the *Query* q. That is, $|q - f_a| = |q - f_b|$, where $f_a < q$ and $f_b > q$. This means that f_a is semantically better than f_b. We calculate the Prepared Matrix PM for f_a and f_b as follows:

$$PM_{f_a} = 1 - \frac{|q - f_a|}{f'_a} \tag{15}$$

We substitute $f'_a = q + |q - f_a|$ in Equation 15 to get

$$PM_{f_a} = 1 - \frac{|q - f_a|}{q + |q - f_a|} \tag{16}$$

Similarly,

$$PM_{f_b} = 1 - \frac{|q - f_b|}{f_b} \tag{17}$$

Since $|q - f_a| = |q - f_b|$ we can substitute f_b with f_a in Equation 17 to get

$$PM_{f_b} = 1 - \frac{|q - f_a|}{q + |q - f_a|} \tag{18}$$

Now, we want to add a boost value BV to the denominator of the PM function of f_a in Equation 17.

$$PM_{f_a} = 1 - \frac{(q - f_a)}{q + (q - f_a) + BV} \tag{19}$$

The value BV added to the denominator should make PM of f_a better than the rank of f_b, yet it must be within the accuracy level acc of the system. That is, it does not exceed the score of the next *Service* $service_c$. That is, $|q - f_a|$ can be at most $|q - f_c| + acc$, where f_c is an attribute in $service_c$.

$$|q - f_a| = |q - f_c| + acc = (f_c - q) + acc \tag{20}$$

where $f_c > q$

Similar to f_a we can calculate PM for f_c by the formula,

$$PM_{f_c} = 1 - \frac{|q - f_c|}{f_c} \tag{21}$$

However, from Equation 20, we can calculate f_c in terms of f_a and say,

$$f_c = q - f_a - acc + q = 2q - f_a - acc \tag{22}$$

Now, we want PM_{f_a} to be more than PM_{f_b} with the help of BV but less than PM_{f_c}, so,

$$PM_{f_b} < PM_{f_a} < PM_{f_c} \tag{23}$$

Thus, from Equation 18, Equation 19 and Equation 21 in Equation 23 we find,

$$1 - \frac{(q - f_a)}{q + (q - f_a)} < 1 - \frac{(q - f_a)}{q + (q - f_a) + BV} < 1 - \frac{(f_c - q)}{f_c}$$

After subtracting 1 from all sides and then multiplying by -1 to get rid of the minus sign (note the change in the direction of the inequality as a result of multiplying by -1) we get

$$\frac{(q-f_a)}{q+(q-f_a)} > \frac{(q-f_a)}{q+(q-f_a)+BV} > \frac{|q-f_c|}{f_c}$$

since we want to solve for BV, we take the reciprocals in the above inequality. We get

$$\frac{q+(q-f_a)}{(q-f_a)} < \frac{q+(q-f_a)+BV}{(q-f_a)} < \frac{f_c}{|q-f_c|}$$

After simplification we get the inequality

$$0 < BV < \frac{f_c*(q-f_a)}{|q-f_c|} - (2q-f_a)$$

We can replace the absolute function $|q-f_c|$ by $f_c - q$, because $f_c > q$. Thus, we arrive at

$$0 < BV < \frac{f_c*(q-f_a)}{(f_c-q)} - (2q-f_a)$$

which can be simplified as follows:

$$i.e., 0 < BV < \frac{qf_c - f_af_c}{(f_c-q)} - (2q-f_a)$$

$$i.e., 0 < BV < \frac{\cancel{qf_c} - \cancel{f_af_c} - \cancel{2}qf_c + \cancel{f_af_c} + 2q^2 - qf_a}{(f_c-q)}$$

Now, from Equation 20, we substitute for f_c

$$0 < BV < \frac{-q(2q-f_a-acc)+2q^2-qf_a}{((2q-f_a-acc)-q)}$$

Hence,

$$0 < BV < \frac{-\cancel{2q^2} + \cancel{qf_a} + q*acc + \cancel{2q^2} - \cancel{qP_a}}{q-f_a-acc}$$

Thus, in order to make an addition to the denominator that will not affect other *Services* within a pre-set accuracy, the value of BV should be

$$0 < BV < \frac{q*acc}{q-f_a-acc} \qquad (24)$$

To make sure that the value of BV is valid with all values of q towards infinity, we calculate the limit of the right side of Equation 24. To avoid indeterminate form $\frac{\infty}{\infty}$ in the limit, we divide throughout by q, $q > 0$.

$$\frac{q * acc * \frac{1}{q}}{(q - f_a - acc) * \frac{1}{q}} = \frac{acc}{1 - \frac{f_a}{q} - \frac{acc}{q}} \tag{25}$$

By applying the limit function on Equation 25, we find,

$$\lim_{q\to\infty} \frac{q * acc * \frac{1}{q}}{(q - f_a - acc) * \frac{1}{q}} = \frac{\lim_{q\to\infty} acc}{\lim_{q\to\infty} 1 - \lim_{q\to\infty} \frac{f_a}{q} - \lim_{q\to\infty} \frac{acc}{q}} = acc$$

As a result, the value BV that would remain valid for all values of q up to infinity should be more than 0 and less than acc. That is any value for BV in the interval $(0, acc)$ may be chosen. Hence, the RC_{Exact} function with semantic preference would become

$$RC_{Exact} = \begin{cases} |-1 + \frac{\triangle}{s' + (BV * X)}| & s < q \\ |+1 - \frac{\triangle}{s}| & s > q \end{cases} \tag{26}$$

where the value of "X" is the same one defined earlier ($+1$ for LB and -1 for MB semantics). This will cause the overall score of the case $s < q$ to increase in case of LB semantic because it is the preferable case, and decrease with MB semantics.

Figure 23 shows how the *BoostValue* is applied in our algorithm. Basically, it is not applied on a particular feature where symmetry occurs, but rather it is applied on all the attributes that fall under the case $s < q$. To clarify, for LB semantics the value *BoostValue* is added to the denominator (because $x = +1$), which in return will increase the overall value of the score. Therefore, all values of all attributes under this case in the LB semantics will receive this privilege. On the other hand, if the semantic is MB, *BoostValue* will be subtracted from the denominator (because $x = -1$) which will result in decreasing the overall score generated for this case, and thus, make the other case, i.e. $s > q$, better.

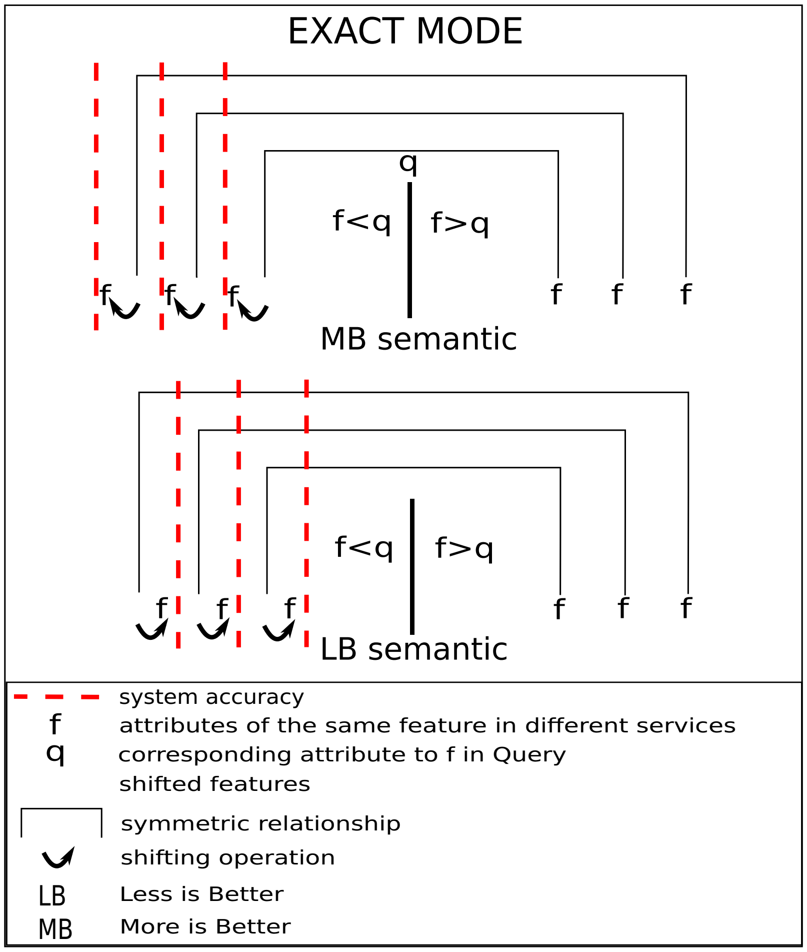


Figure 23: The way of applying the *BoostValue* on the properties based on the semantics when *Exact Mode* is ON

Example 6. *In this example, we illustrate the symmetry and semantic preference principles. We perform those concepts for both semantics, MB and LB in Exact Mod. The user request is shown in Table 14. The available* Services *are shown in Table 15.*

-	$Price(f_1)$	$Reliability(f_2)$
$Semantic$	$LB(X=1)$	$MB(X=-1)$
$Ranking\ Mode$	*Exact Mode*	*Exact Mode*
$Query$	200$	50%
$System\ Accuracy(acc)$	1	

Table 14: User Request

We have chosen the values of service attributes such that they are ordered according to their closeness to the corresponding values defined in Query.

$$price_{S_3} \ll price_{S_4} \ll price_{S_1} \& price_{S_2}$$

and,

$$reliability_{S_2} \& reliability_{S_1} \ll reliability_{S_4} \ll reliability_{S_3}$$

where the symbol $\ll$ *means "*$better\ than$*" and the symbol* & *means "*$similar\ quality$*".*

-	$Price(f_1)$	$Reliability(f_2)$
$Service_1(S_1)$	$150\$(s < q)$	$90\%(s > q)$
$Service_2(S_2)$	$250\$(s > q)$	$10\%(s < q)$
$Service_3(S_3)$	$249\$(s > q)$	$91\%(s > q)$
$Service_4(S_4)$	$249.9\$(s > q)$	$90.1\%(s > q)$

Table 15: Available Services

We have applied the Similarity Measure, RC_{Exact} *(Equation 26), considering the semantic preference with a system accuracy of* 1, $(BV = 1)$ *as defined in user request. Then, we have recorded the results in Table 16.*

$Services$	$Price$	$Reliability$
S_1	0.8007	0.5556
S_2	0.8	0.5506
S_3	0.803	0.5495
S_4	0.8003	0.5549

Table 16: Scores with $acc = 1$

By looking at the results for each feature of available Services, we find that the symmetry in price features between $Price_{S_1}$ *and* $Price_{S_2}$ *has been broken correctly. That is, the semantically better(cheaper) value (*$Price_{S_1}$*) has received a higher score. However, the features* $Price_{S_3}$ *and* $Price_{S_4}$ *provide closer values to the price feature defined in Query. So, they should receive higher scores than* $Price_{S_1}$ *and* $Price_{S_2}$ *even after boosting the score of* S_1 *for being semantically better. Indeed,* $Price_{S_3}$ *received higher score, but not* $Price_{S_4}$*. This is because the difference between*

$Price_{S_4}$ and the symmetric feature $Price_{S_2}$ is less than the system accuracy, i.e. $acc = 1$, while the difference between $Price_{S_3}$ and the symmetric feature is equal to system accuracy which makes it preserves its position.

Similarly, in $Reliability$ feature Services S_1 and S_2 are symmetric. Based on the definition of Semantic Preference, BoostValue is to be applied only for the case $Feature_{Service} < Feature_{Query}$ either by subtracting or adding it to the denominator. Because reliability attribute belongs to MB semantics subtracting the denominator by BoostValue in the case $Feature_{Service} < Feature_{Query}$ should be done. As a result, the symmetry between $Reliability_{S_1}$ and $Reliability_{S_2}$ is broken in favour of the the feature $Reliability_{S_1}$. However, this subtraction made the attribute $Reliability_{S_2}$ become worse than $Reliability_{S_4}$ although $Reliability_{S_2}$ is closer to Reliability defined in Query than $Reliability_{S_4}$ by 0.1. Similar to $Price$, the inaccuracy of the result is because the system accuracy is set to 1, thus the Semantic Preference will not consider cases for differences between Services less than 1.

To correct the results shown above, we set the accuracy to $acc = 0.1$, we obtain the scores recorded in Table 17. Note how score of $Price_{S_1}$ has become better than the score of $Price_{S_2}$, yet it remains less than the score of $Price_{S_4}$. Similarly, the score of $Reliability_{S_2}$ has become less than the score of $Reliability_{S_1}$, yet it remains higher than the score of $Reliability_{S_4}$.

$Services$	$Price$	$Reliability$
S_1	0.8001	0.5556
S_2	0.8	0.5551
S_3	0.803	0.5495
S_4	0.8003	0.5549

Table 17: Results of scores for the features of the available Services with ”$acc = 0.1$”

5.4 Scaling Method

Scaling is a function f that change a value x in a finite interval (a,b) to a value $x' = f(x)$ in another interval $[A,B]$. The function f satisfies the property that if $x < x'$ in $[a,b]$ then $f(x) < f(x')$, $\frac{x}{x'} = \frac{f(x)}{f(x')}$.

The scaling function that we use is a linear transformation based on the geometry of a line segment [Tek06].

$$nv = \frac{(nmax - nmin)(v - cmin)}{(cmax - cmin)} + nmin \tag{27}$$

where $v \in [cmin, cmax]$, and $nv \in [nmin, nmax]$.

Basically, there is no need for scaling when we use the original *Similarity Measure* RC. However, because we have introduced new characteristics to this *Similarity Measure*, such as semantics and symmetry, the generated scores have been perturbed. Fair ranking should be consistent, as defined in Chapter 2. The ranks produced by the *Similarity Measure* SM_X should be as expected by the user's expressed requirements. So, penalties and rewards are to be produced based on the difference between the *Query*'s attributes and the *Services*' attributes. In any semantics, if a *Service* attribute (s) differs from the same *Query* attribute (q) by an amount a, the $penalty$ or $reward$ assigned to s should always be a function of a. That is, it is independent of *Query* or *Service* but depends only on the deviation between the values of their attributes in each semantics.

Example 7. *We use the same notation as before to denote by q a* Query *attribute value and s to denote the corresponding* Service *attribute value. Suppose $q = 10$. We apply the* Similarity Measure *SM_X (defined in Equation 11 in Section 5.2) on different* Services*, given by their s values $s = [0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30]$. With each value of s we consider all semantics and modes, except EB semantics for which the range is already defined in the previous section.*

Table 18 shows that the penalty produced by SM_X for $s = 12$ in LB semantics is equal to the

Modes	*Best Mode*				*Exact Mode*		ANY
Type	*LB*		*MB*		ANY		*EB*
Case / S	$s < q$	$s > q$	$s < q$	$s > q$	$s < q$	$s > q$	ANY
score type	$reward$	$penalty$	$penalty$	$reward$	$penalty$	$penalty$	ANY
0	$1+0.50$	-	0.50	-	0.50	-	-
2	$1+0.44$	-	0.56	-	0.56	-	-
4	$1+0.38$	-	0.63	-	0.63	-	-
6	$1+0.29$	-	0.71	-	0.71	-	-
8	$1+0.17$	-	0.83	-	0.83	-	-
10	$1(reward_{min})$		$1(reward_{min})$		$1\ (reward)$		-
12	-	0.83	-	$1+0.17$	-	0.83	-
14	-	0.71	-	$1+0.29$	-	0.71	-
16	-	0.63	-	$1+0.38$	-	0.63	-
18	-	0.56	-	$1+0.44$	-	0.56	-
20	-	0.50	-	$1+0.50$	-	0.50	-
22	-	0.45	-	$1+0.55$	-	0.45	-
24	-	0.42	-	$1+0.58$	-	0.42	-
26	-	0.38	-	$1+0.62$	-	0.38	-
28	-	0.36	-	$1+0.64$	-	0.36	-
30	-	0.33	-	$1+0.67$	-	0.33	-
$\vdots$	-	$\vdots$	-	$\vdots$	-	$\vdots$	-
$+\infty$	-	0	-	2	-	1	-
$s=(0,\infty)$	$(1,1.5)$	$(0,1)$	$(0.5,1)$	$(1,2)$	$(0,1)$	$(0,1)$	$(0,1)$

Table 18: Motivation to use Scaling for MB and LB Semantics.

one produced for $s = 8$ in MB semantics. This implies that their penalties are consistent. However, the range of penalties produced in LB is $(0, 1)$, while the range of penalties for MB is $(0.5, 1)$. This is normal because the values of s lie in the interval $(0, +\infty)$. That is, we don't consider negative values in the inputs which makes the lower bound of MB penalties stops at 0.5. As a result, there has to be a difference in the ranges. However, the range $(0.5, 1)$ is very tight. Our example is using small values of s and q. However, if bigger values are used the tightness of the range will become clearer, and the difference between the produced results becomes high in precision. To illustrate this point, consider the values $s = [99, 98]$, $q = 100$ with MB semantics. Since $s < q$ for both values of s, a penalty should be assigned. When $s = 99$ the assigned penalty is $penalty = 0.99497$, while for $s = 98$ the assigned penalty is $penalty = 0.989899$. Hence the difference, $diff_1 = 0.0051$, *between the two penalties is very small. We solve this issue by widening the range* $(0.5, 1)$ *to* $(0, 1)$ *by using the Scaling Method introduced earlier in this Section. After scaling, the penalties are* $penalty = 0.98994$ *for* $s = 99$ *and* $penalty = 0.979798$ *for* $s = 98$*. The difference between penalties is* $diff_2 = 0.01014$*, which is an improvement by* $\frac{(diff_2 - diff_1)}{diff_1} * 100 = 98.82\%$*. As explained earlier, because of the difference in the input range between* LB *and* MB *in the penalty zone, and to produce similar penalties, we should scale the values of* LB *to the range* $(-1, 1)$*. Hence, the penalties in* MB *semantics stop at* 0 *when inputs stop at* 0*, while the penalties of* LB *continue to* -1 *as inputs approach* $+\infty$*.*

For assigning the $rewards$ *we follow a similar procedure. The* $rewards$ *in the semantics* LB *will stop at* 2 *when inputs reach "*0*", while the* $rewards$ *in* MB *semantics continue to "*3*" as inputs approach* $+\infty$*.*

As for Exact Mode*, it is a different concept. This is because the* $reward$ *in this mode is assigned in only one instance, when the value of* q *matches the value of* s*. For all other values of* s *we assign* $penalties$*. Similar to* Best Mode*, a scaling method is required to stretch the scores up to the range* $(0, 1)$ *for the case* $s < q$ *and* $(-1, 1)$ *for the case* $s > q$*.*

Thus, by applying the aforementioned changes on the produced scores we obtain the scaled scores shown in Table 19

Remark. *Although there are still differences in the final upper/lower bound of the range for some semantics, it is still considered fair, as the scores are evenly produced for all semantics. Also, the different bounds of the range are based on the input range.*

Modes	*Best Mode*				*Exact Mode*		ANY
Type	*LB*		*MB*		ANY		*EB*
Case / S	$s < q$	$s > q$	$s < q$	$s > q$	$s < q$	$s > q$	ANY
score type	*reward*	*penalty*	*penalty*	*reward*	*penalty*	*penalty*	ANY
0	2	-	0	-	0	-	-
2	1.88	-	0.1	-	0.1	-	-
4	1.76	-	0.26	-	0.26	-	-
6	1.58	-	0.44	-	0.44	-	-
8	1.34	-	0.66	-	0.66	-	-
10	1		1		1		-
12	-	0.66	-	1.34	-	0.66	-
14	-	0.44	-	1.58	-	0.44	-
16	-	0.26	-	1.76	-	0.26	-
18	-	0.1	-	1.88	-	0.1	-
20	-	0	-	2	-	0	-
22	-	-0.1	-	2.1	-	-0.1	-
24	-	-0.16	-	2.16	-	-0.16	-
26	-	-0.24	-	2.24	-	-0.24	-
28	-	-0.28	-	2.28	-	-0.28	-
30	-	-0.34	-	2.34	-	-0.34	-
$\vdots$	-	$\vdots$	-	$\vdots$	-	$\vdots$	-
$+\infty$	-	-1	-	3	-	-1	-
$s = (0, \infty)$	$(1, 2)$	$(-1, 1)$	$(0, 1)$	$(1, 3)$	$(0, 1)$	$(-1, 1)$	$(0, 1)$

Table 19: Scaled Scores in Different Semantics.

5.5 Essential Option

Non-negotiable requirements are common in real life. When we order a *Service*, it is possible that we have some requirements that we are not willing to compromise. It can be with respect to one or more features. Therefore, we incorporate the *Essential Option* feature for ranking *Services*.

Essential Option is a feature-option, so, it can be set for each feature independently. If a feature is not marked essential then it is regarded as not essential. We do not change the *Similarity Measure* to include Essential Option, instead, we use appropriate weights to discriminate and order essential features. Technically, when Essential Option is set for one feature, this feature would have two weights, one is the *Regular Weight* reflecting user preference for this feature and the other *Essential Weight* which reflects that this feature is essential. The *Regular Weight* has been defined earlier in this Chapter. We discuss how to set the *Essential Weight* for an essential features that will make the matching *Service* to be ranked higher in the list.

Both regular and essential weights are vectors having the same length as the *Query*, such that each feature in *Query* has the corresponding weights in these vectors. In the essential weight vector W_E the weight is set to zero for non-essential features, and is set to w_e for an essential feature. Considering W_R denotes the regular weight vector, the weight vector for a *Query* is $W = W_R + W_E$, where $+$ denotes vector addition. The criterion for choosing the essential weight(s) is that their choice will outweigh other features no matter how good or bad they were. Below we explain how to calculate the essential weights.

Let N denotes the number of features in a *Query*. For the sake of clarity and simplicity, assume that the first feature in the *Query* is an essential feature (q_E) and the others are regular and non-essential features q_i. That is,

$$Query = \{q_{1E}, q_2, q_3, q_4, \ldots, q_N\}$$

Let the scores for an available *Service* be defined as $scores = (x_1, x_2, \ldots, x_N)$, where x_i is the i_{th} score of the i_{th} feature defined in *Service*. The total rank TR of a *Service* is the weighted sum of scores. That is,

$$TR = \sum_{i=1}^{N} w_i * x_i,$$

where w_i is the i^{th} element of the weights vector W. We rewrite this sum as

$$TR = WS_E + WS_R \tag{28}$$

where, WS_E is the weighted score for all scores of services corresponding to the essential feature in *Query*and WS_R is the weighted score for all non-essential features

In our example, there is only one essential feature. We want to calculate w_e for this essential feature, allowing all other non-essential features to assume maximum possible weight w_h. The rationale is that such a value w_e will serve as upper bound for essential weights when the weights of other non-essential features are less than w_e.

$$WS_E = (w_e + w_h) * x_1 \tag{29}$$

and

$$WS_R = w_h * x_2 + w_h * x_3 + \cdots + w_h * x_N \tag{30}$$

Now, we want to define a worst case scenario for which we can obtain a value of w_e that prefers a total rank over another just based on the essential feature, no matter how good or bad other non-features are. So, we consider two *Services*, one which has all features in worst case (highest penalty) and one feature which is essential in best case (highest reward), and another that has all features in best case (highest reward) and only one essential feature which is “slightly worse” than the one defined for the first *Service*. The intent is to rank the first *Service* (best essential feature) on top of second *Service* (worse essential feature), although it has all the good non-essential values in it. Let a_i and b_i denote scores of the i_{th} feature in the *Services*.

$$Service_1 = [a1_{bestE}, a2_{worst}, a3_{worst}, \ldots, aN_{worst}] \text{ where } a2 = a3 = \cdots = aN \tag{31}$$

$$Service_2 = [b1_{worseE}, b2_{best}, b3_{best}, \ldots, bN_{best}] \; where \; b2 = b3 = \cdots = bN \tag{32}$$

We want the total rank TR_1 of *Service*$_1$ to be higher than the total rank TR_2 of *Service*$_2$, because $a1_{bestE} > b1_{worseE}$ regardless of other non-essential features.

$$TR_1 > TR_2 \tag{33}$$

From Equation 28,Equation 29, and Equation 30, we can say,

$TR_1 = WS_E1 + WS_R1$

where

$WS_E1 = a1_{bestE} * (w_e + w_h)$

$WS_R1 = a2_{worst} * (w_h) + \cdots + aN_{worst} * (w_h) = w_h * (N-1) * a2_{worst}$

Thus,

$$TR_1 = (w_e + w_h)a1_{bestE} + w_h * (N-1) * a2_{worst} \tag{34}$$

Similarly,

$TR_2 = WS_E2 + WS_R2$

where

$WS_E2 = b2_{worseE} * (w_e + w_h)$

$WS_R2 = b2_{best} * (w_h) + \cdots + bN_{best} * (w_h) = w_h * (N-1) * b2_{best}$

Thus,

$$TR_2 = (w_e + w_h)b1_{worseE} + w_h * (N-1) * b2_{best} \tag{35}$$

Hence, from Equation 34 and Equation 35 in Equation 33, we can conclude,

$$(w_e + w_h)a1_{bestE} + w_h * (N-1) * a2_{worst} > (w_e + w_h)b1_{worseE} + w_h * (N-1) * b2_{best} \tag{36}$$

By solving the above inequality as an attempt to obtain the value of w_e we find,

$$w_e > w_h \frac{(a1_{bestE} - b1_{worseE} + (N-1)(b2_{best} - a2_{worst}))}{a1_{bestE} - b1_{worseE}} \tag{37}$$

The values for all the variables in Equation 37 are known, except for the variable $b_1worseE$. That is, we know the highest reward is 3, and the highest penalty is -2. Also, we know the highest weight. Hence, $a1_{bestE} = 3$, $b2_{best} = 3$, $a2_{worst} = -2$ and $w_h = 0.005$. The difference between the two essential features is the accuracy of this option, which should be defined by the system or users as an input in the request. How accurate should this essential option be?

Specifically, the difference between the two essential features scores should be such that it is possible to prefer one *Service* over the other. For example, if the score of an essential feature in a $Service_a$ is $a = 2$, and the score f the same feature in another $Service_b$ is $b = 1.999$, the difference between the two scores is $diff = a - b = 0.001$. If the accuracy of the essential option is set to 0.001 it will prefer $Service_a$ over $Service_b$ because the difference between the scores of essential features is less than or equal to the accuracy of the algorithm. However, if the accuracy of the algorithm is set to 0.01, the algorithm is not guaranteed to prefer $Service_a$ over $Service_b$. Hence, we introduce the user-defined (or system-defined) essential accuracy parameter $EssAcc = a1_{bestE} - b1_{worseE}$ in Equation 37 to get

$$w_e > w_h \frac{(EssAcc + (N-1)(b2_{best} - a2_{worst}))}{EssAcc} \tag{38}$$

We add 1 to the right side of Equation 38 to choose w_e:

$$w_e = w_h \frac{(EssAcc + (N-1)(b2_{best} - a2_{worst}))}{EssAcc} + 1 \tag{39}$$

Remark. *The small numbers we defined as our weights helped us in obtaining the essential weight as a manageable value. Otherwise, w_e will be a large number.*

At this point, we have obtained the value of w_e that can make the essential feature becomes the pivot of the ranking regardless of how good or bad are the other non-essential features.

Next, we add one more flexibility for this option to allow users specify the level of importance to essential features, when more than one feature is essential. This is performed using the regular weight. That is, we multiply the essential weight w_e by the regular weight w_r assigned to each feature. Hence, we create different instances of the essential w_e based on the values of w_r assigned for each feature. However, since the regular weights are defined as fractions (divided by 1000), it will make the value of w_e smaller when multiplication is performed, which might make the value of w_e smaller than the sufficient value to prefer *Services* based on essential features. Thus, we avoid the fraction of w_r by multiplying it by 1000 and then multiply w_r by w_e. That is,

$$w_i = w_{ri} + (w_{ei} * w_{ri} * 1000),$$

where, w_i is the i_{th} weight in the Weight vector defined in correspondence to the i_{th} attribute defined in *Query* vector.

Once the weight vector W is calculated and the Prepared Matrix (PM) of scores (for features against available *Services*) is computed, the total ranking vector TR is calculated with the formula

$$TR = W \times PM$$

Example 8. *This example illustrates how essential option is to be treated in* Service *ranking. The essential option is applied on scores that have already been calculated. So, we assume that this phase is already performed, and we have the Prepared Matrix(PM) that contains the scores of the* Services *features as shown in the Table below. Let $Feature_{1E}$ be the only essential feature. So, the*

-	S_1	S_2	S_3	S_4
$Feature_{1E}$	3	2.99	2.9	2.8
$Feature_2$	-2	3	3	-2
$Feature_3$	-2	3	3	-2
$Feature_4$	-2	3	3	-2

user is interested to get the Service *with maximum score of the essential feature no matter how good the other non-essential features are in other* Services. *Let us assume that EssAcc* $= 0.1$. *Since $Feature_{1E}$ is what matters, it is easy to recognize that the ranking should be:*

$$S_1 \ll S_2 \ll S_3 \ll S_4, \textit{ where } \ll \textit{ means "better than"}$$

We use the values $N = 4$ for number of features, $worst = -2$ for the worst value (shown in the table), and $best = 3$ for the best value. The highest weight w_h defined in our algorithm is $w_h = 0.005$. Using these values in Equation 39 *we calculate the value of w_e for $Feature_{1E}$.*

$$w_e = 1.755$$

Since all other features are non-essential, the vector W_E is defined as follows:

$$W_E = [1.755, 0, 0, 0]$$

Let all features have the highest importance w_h. So, the regular weight vector is

$$W_R = [0.005, 0.005, 0.005, 0.005]$$

Since there is only one essential feature, we do not multiply the regular weight by 1000. *Hence, the total weight vector W is*

$$W = W_R + W_E = [1.76, 0.005, 0.005, 0.005]$$

The product $TR = W \times PM$ is the vector shown below.

-	S_1	S_2	S_3	S_4
TR	5.25	5.3074	5.149	4.898

The above table clearly shows the following order of Service

$$S_2 \ll S_1 \ll S_3 \ll S_4 \text{ where the symbol } \ll \text{ means “better than”}$$

This result is against the expectation of essential option definition. Although $Feature_{1E}$ of S_1 is higher than $Feature_{1E}$ of S_2, the total rank for S_2 is higher. This is because the difference between the values of these fatures is $diff = 0.01$, which is smaller than the Essential Accuracy, i.e. EssAcc $= 0.1$. Thus, by changing the value of EssAcc to 0.01, we can obtain more accurate results. We recalculate w_e using the new value of EssAcc, we get $w_e = 8.505 + 0.005 = 8.51$. Then, using the new value of w_e, we recompute W and $W \times PR$, and obtain the following results,

-	S_1	S_2	S_3	S_4
TR	25.5	25.4899	24.724	23.798

The new order of the Services becomes,

$$S_1 \ll S_2 \ll S_3 \ll S_4 \text{ where the symbol } \ll \text{ means “better than”}$$

which clearly shows that S_1 has a higher total score than S_2.

Remark. *There are two valid methods to determine the the essential mode accuracy. It can be assigned by users at building request time, or can be pre-set based on the application domain. The former, makes the algorithm more user-based, and adds more flexibility, dynamism and control to users. However, the problem of this choice that users might not understand how the accuracy works. This is because accuracy is at scores level not at inputs level. This require users to comprehensively understand how scores are generated. As a consequence, users might misuse it and receive unexpected results. The latter, however, makes the algorithm more controlled and easier to understand by users, but limits the users' control of the output. We prefer the latter to comply with our claim on offering a user-friendly and understandable requesting concept.*

5.6 All Best Option

For a given *Query*, assume that scores has been assigned to the different attributes of *Services*. In practice, these scores are simply added together to produce a total score. In this process, features with highest scores may hide the effect of the lower scores, because the total score is acceptable to the user. This effect of hiding is called *compensation*. The goal of AllBest Option is to minimize the effect of compensation and to rank *Services* which match all *Query* features (at their minimum requirements) as implied in either *Exact Mode* or *Best Mode* higher in the final ranking.

The option *AllBest* can only be set for the entire *Query*, unlike other options that can be set for each feature. Therefore, the change caused by this option is applied on the total ranks produced by the algorithm rather than the scores generated by the *Similarity Measure*. We propose a method that boosts the rank of each *Service* based on how close each feature is to the *Query*. This method performs calculations aside from the calculations of scores of each feature, yet in parallel. It records its results in a matrix called *AllBest Matrix(ABM)* that has a size equals to the size of the matrix PM that holds all the scores of all features. We define the matrix ABM as follows:

$$ABM[i,j] = \begin{cases} 1 & \text{if } PM[i,j] \text{ is a } reward \\ PM[i,j] & \text{if } PM[i,j] \text{ is a } penalty \end{cases} \tag{40}$$

The rationale for assigning 1 is that it is the minimum $reward$. At the same time we do not want to increase the $penalty$. This way, we minimize the effect of compensation.

Eventually, ABM matrix will contain columns that represent *Services*, and rows that represent features of these *Services*. Each column of ABM matrix is aggregated to form what we call the *AllBest Vector(ABV)* such that ABV_i is the $AllBest\ Value$ for the i_{th} *Service*. The value ABV_i of a column is to be added to the final rank of the corresponding *Service*. Formally, we construct the Vector ABV for the *Services* as follow:

$$ABV[j] = \sum_{i=1}^{N} ABM[i,j]$$

where, $ABV[j]$ is the ABV value for the j_{th} *Service* in PM matrix. We define the total rank for $Service_j$ as $TR'[j] = TR[j] + ABV[j]$. We illustrate $AllBest$ method with a simple example shown in Figure 24

In the following subsection we mathematically determine the value ABV for a *Service* that can make the rank of that *Service* higher than other *Services* whose ABV values are smaller.

5.6.1 The AllBest Value

Let N denote the number of features in a *Query* Q, and F_{set} is the set of features defined in a *Query* and *Services*.

$$F_{set} = \{f_1, f_2, f_3, f_4, \ldots, f_N\}$$

In order to obtain a value of ABV that can work with any case, we should define a worst case and examine our ABV with it. the worst case scenario is given by the Service $S_{good} =$

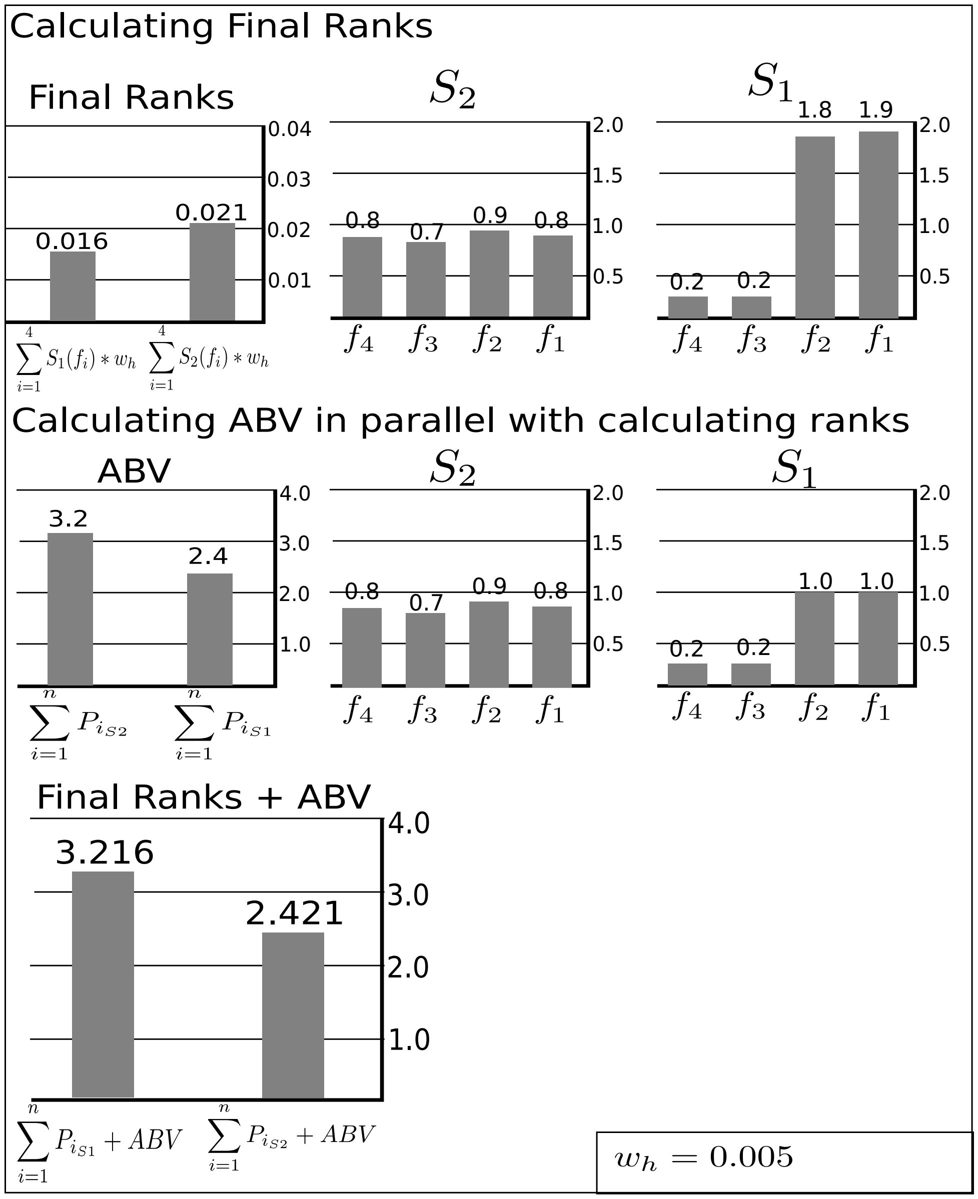


Figure 24: Simple example that illustrates *AllBest* method

$\{f_{1bad}, f_{2good}, \ldots, f_{Ngood}\}$, where f_{1bad} means that the minimum requirement is not met by that attribute and f_{igood}, $i \geq 2$ means that the i_{th} feature gets the highest score. We call this the worst case because the effect of compensation is fully met here. Let $S_{min} = \{f_{1min}, f_{2min}, \ldots, f_{Nmin}\}$, be the Service in which every feature defined in *Query* is met minimally (according to mode). The goal of all best algorithm is to rank S_{min} higher than S_{good}.

We conduct a mathematical analysis to justify that consistency is preserved by the choice of

ABV values. The ABV value for TR_{min} (total weight corresponding to S_{min}) is N, and the ABV value for TR_{good} (total weight corresponding to S_{good}), is $N-1+y$ where y is the value f_{1bad}. In order to rank S_{min} higher than S_{good}, we want

$$TR_{min} + N > TR_{good} + y + N - 1 \tag{41}$$

This is rewritten as,

$$TR_{min} + x > TR_{good} \tag{42}$$

where $x = 1 - y$ and $0 < x < 1$. Let w_h be the maximum weight that a user can assign. So the total rank for S_{good} is

$$TR_{good} = w_h * f_{1bad} + w_h * f_{2good} + w_h * f_{3good} + \cdots + w_h * f_{Ngood}$$

$$= TR_{good} = w_h * f_{1bad} + w_h * \sum(f_{2good} + f_{3good} + \cdots + f_{Ngood})$$

Hence,

$$TR_{good} = w_h * f_{1bad} + w_h * (N-1) * f_{igood}, 1 < i \leq N \tag{43}$$

We calculate TR_{min}:

$$TR_{min} = w_h * N * f_{jmin}, 0 < j \leq N \tag{44}$$

Thus, from Equation 43 and Equation 44 in Equation 42

$$w_h * N * f_{jmin} + x > w_h * f_{1bad} + w_h * (N-1) * f_{igood} \tag{45}$$

We substitute the values of $f_{jmin} = 1$, $f_{bad} = 0.9$ and $f_{igood} = 3$ in Equation 45 to get,

$$w_h * N * 1 + x > w_h * (3N - 3 + 0.9)$$

Hence,

$$w_h * N + x > 3w_h * N - 2.1w_h$$

That is,

$$x > w_h * N - 2.1w_h \tag{46}$$

i.e,

$$1 - y > 2w_h N - 10.5 \times 10^{-3}$$

Which means,

$$1 - (2w_h * N - 10.5 \times 10^{-3}) > y$$

Hence,

$$y < 1 - 2w_h * N + 10.5 \times 10^{-3} \tag{47}$$

Substituting $w_h = 0.005$ in Equation 47 we get,

$$y < 1 - 0.01N - 10.5 \times 10^{-3}$$

if $N > 100$ then $2w_h * N > 1$ this means that there is a possibility for y becoming negative for values $N > 100$. For values $N <= 100$,the value of y lies between 0 and 1. In practical user-centric applications, we do not expect any *Query* or *Service* to have 100 features. That is, for all practical applications the *ABV* values that we have chosen are consistent with our definition of maximum/minimum penalty. Next, we give a comprehensive example to illustrate all best option.

Example 9. *Let a User Request Q be defined as in Table 20 considering that AllBest option is ON,i.e AllBest=1. Assume that the available Services are defined as in Table 21*

	Feature 1	Feature 2	Feature 3
Query	12000	10000	400
Mode	Best	Best	Exact
Semantics	LB	LB	LB
ESS	0	0	0
AllBest	**1**		

Table 20: User Request

	Feature 1	Feature 2	Feature 3
Service1	1	1	700
Service2	12000	10000	400
Service3	12000	10000	440
Service4	12100	11000	420

Table 21: available services

Calculating the Prepared Matrix (PR) along with the ABM matrix, we come with the following two matrices

$$PR = \begin{array}{c|cccc} & S_{1\downarrow} & S_{2\downarrow} & S_{3\downarrow} & S_{4\downarrow} \\ f_1 \rightarrow & 2.00 & 1.00 & 1.00 & 0.99 \\ f_2 \rightarrow & 2.00 & 1.00 & 1.00 & 0.91 \\ f_3 \rightarrow & 0.57 & 1.00 & 0.91 & 0.95 \end{array} \quad ABM = \begin{array}{cccc} S_{1\downarrow} & S_{2\downarrow} & S_{3\downarrow} & S_{4\downarrow} \\ 1 & 1 & 1 & 0.99 \\ 1 & 1 & 1 & 0.91 \\ 0.57 & 1 & 0.91 & 0.95 \end{array} \tag{48}$$

We compute vectors TR and ABV by assuming $w_h = 0.005$,

$$PR = \begin{pmatrix} 0.0229 & 0.015 & 0.0146 & 0.0143 \end{pmatrix} \quad ABV = \begin{pmatrix} 2.57 & 3 & 2.91 & 2.85 \end{pmatrix} \tag{49}$$

Before adding this value to the total rank (TR) of Services*, the ranking list would look as follows:*

$$TR_{S1} = 0.0229$$

$$TR_{S2} = 0.015$$

$$TR_{S3} = 0.0146$$

$$TR_{S4} = 0.0143$$

That is, the Service ranking is $S1 > S2 > S3 > S4$. This means that due to the high quality of some of the properties of Service $S1$, they compensated the shortcoming of one bad feature. This is acceptable; However, in some cases users don't want to see this compensation.

Thus, by applying the final step of the $AllBest$ Method, we get the total ranks:

$$TR_{S1} = 0.0229 + 2.57 = 2.5929$$

$$TR_{S2} = 0.015 + 3 = 3.015$$

$$TR_{S3} = 0.0146 + 2.91 = 2.9246$$

$$TR_{S4} = 0.0143 + 2.85 = 2.8643$$

Hence the ranking of Services after adding the ABV values to final ranks becomes as shown in Table 22

Order	Services
1	S_2
2	S_3
3	S_4
4	S_1

Table 22: Results after applying $AllBest$ Method.

Note that this method does not remove compensation, but keeps it to minimal.

5.7 Range Option

In real life, looking for *Services* in a specific price range is common. For example, we tend to tell shop attendants our budget, so that they provide us with products they have within our desired price. Also, they start providing us with the least price within the specified price range with respect to other features of the product. We would like to integrate the same facility in online *Services* consumption. This is through offering the *Range Option*.

Technically, range option can be integrated in the X-Algorithm by making attributes of *Services* within the specified range in *Query* receive highest scores and attributes outside the range receive less scores, gradually as decreasing as they go further away from the specified range. However, this would mean that all values within the range are treated at the same level of quality, regardless of their semantics. For example, suppose a *Query* contains a price feature

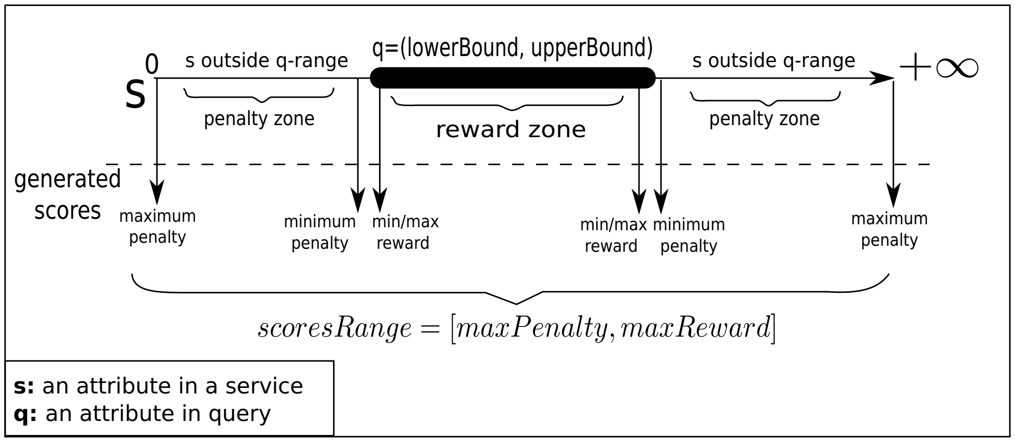


Figure 25: The assignment of different score types with Range Option

defined as the range $q = (1000, 5000)$. Now, is it fair to rank attributes of *Services* of price 1000 similar to those of the price 5000? if we want to give consumers options they would have in real life, the answer is No. This is because even within the specified range, semantics count. For *LB* semantics, 1000 should receive a higher score than 5000. This means that the values of attributes within the specified range should receive a range of $reward$ based on semantics. Otherwise, $penalty$ should be assigned. The value of $penalty$ ranges from $minPenalty$ to $maxPenalty$ based on how far the value of the attribute is from the specified range. Thus, for *LB* semantics we assign $maxReward$ when the price feature of a *Service* equals to the least value of the price range specified in *Query*, and for *MB* semantics the $maxReward$ should be assigned when the price feature of a *Service* equals to the maximum value of the price range specified in *Query*. See Figure 25. Therefore, we need to perform ranking inside the range as well as ranking outside the range.

The ranking inside the range should be done with respect to semantics specified in *Query*. Thus, it is always going to be functioning as in *Best Mode*. That is, based on the definition of "better", the values within the range are ranked. In order to achieve "better" for *MB* semantics and *LB* semantics we need to fix a reference point. We consider the mid point of the range

$[a, b]$ specified in the *Query* as the $Reference\ Value(ref)$:

$$ref = \frac{(a+b)}{2} \tag{50}$$

This reference value preserves all previous characteristics defined for the X-Algorithm such as symmetry and consistency. We need to prefer values of *Service* attributes that are within the required range specified in a *Query*, and at the same time consider other features of the *Service* with respect to the specified semantics, yet eliminate those *Services* that provide values outside the specified range in a *Query*. So, we calculate $reward$ as well as $penalty$ with respect to ref within the range, as explained earlier. Recall that the range for penalties of our algorithm is $(-1, 1)$, and it only turns to negative when $|s - q| > q$ as shown in Table 19 in Section 5.4. Therefore, defining the value ref as *Query* helps in discriminating between the penalties generated for values inside the range and outside the range. That is, because of the natural relationship between the reference value and the lower/upper bound of the range ($|v - ref| < ref$ where $v \in (lowerBound, upperBound)$), the $penalty$ range produced within the range is always in the range $(0, maxReward)$. To penalize the values outside the range with a non-overlapping values with penalties inside the range, we introduce another function $Out\ Of\ Range\ Penalty\ Function(OPF)$ for calculating penalties outside the range with respect to the same reference value ref. This function is similar to the RC_X function, yet it produces negative numbers. These negative numbers decrease in absolute value as the value of a *Service* attribute gets further away from the range specified in the *Query*. The penalties calculated by OPF lie in the range $(-1, 0)$, which is non-overlapping with the range $(0, maxReward)$. Since the function OPF performs the calculations of penalties with respect to one value ref, it considers semantics and symmetry in the same way as RC_{Exact}. To clarify, the values outside the range are scored based on their closeness to the range specified in *Query*.

Based on the above discussion, we come up with the following OPF function.

$$OPF = \begin{cases} -\frac{s-ref}{s} & s > q_{upper} \\ -\frac{ref-s}{s'+(BV*X)} & s < q_{lower} \end{cases} \tag{51}$$

where, q_{upper} and q_{lower} are respectively the upper and lower bounds defined in *Query* range for a specific feature, s is the value of the same feature in a *Service*, $s' = |ref - S| + ref$, $ref = \frac{q_{upper}+q_{lower}}{2}$, BV is *BoostValue* and $X = -1$ for MB and $X = +1$ for LB semantics.

Example 10. *This example illustrates Service ranking for a range Query, for the semantics LB and MB. Let $q = [40, 70]$. Thus, $q_{lower} = 40$ and $q_{upper} = 70$. Let $s_1 = 30$, $s_2 = 80$, $s_3 = 50$, and $s_4 = 60$ be the values for the same feature (as in Query) in four different Services. The reference value $ref = (40+70)/2 = 55$*

Case 1: LB semantics. The following calculations are done with $X = 1$, and $BV = 1$.

Since, $s_1 < Q_{lower}$ and $s_2 > Q_{upper}$, both lie outside the range. So, penalties using OPF are calculated.

$$s_1' = ref + (ref - s_1) = 55 + (55 - 30) = 80$$

Using the OPF function in Equation 51 we get the penalty for $s_1 (s_1 < Q_{lower})$.

$$OPF_{s_1} = -\frac{s_1' - ref}{s_1' + (BV * X)} = \frac{80-55}{80+1} = -0.3086$$

We calculate the penalty OPF for $s_2 (s_2 > Q_{upper})$

$$OPF_{s_2} = -\frac{s_2 - ref}{s_2} = -0.3125$$

Since the features s_3 and s_4 are within the range, they are calculated regularly as introduced before. Thus,

$$s_3 = |X + \frac{ref - s_3}{s_3' + (BV * X)}| = 1.082$$

$$s_4 = |X - \frac{s_4 - ref}{s_4}| = 0.917$$

Notice that the penalty in s_2 is greater in absolute value than in s_1. This is because s_1 is semantically better than s_2 (cheaper). Also, note how s_3 received higher score than s_4 for being cheaper as well.

Case 2: MB semantics. $X = -1$. calculate s_1' for the case $s < Q_{lower}$

$$s_1' = ref + (ref - s_1) = 55 + (55 - 30) = 80$$

Using the OPF function 51 we get the penalty for s_1.

$$OPF_{s_1} = -\frac{s_1' - ref}{s_1' + (BV * X)} = -\frac{80 - 55}{80 - 1} = -0.3165$$

We calculate the penalty for s_2

$$OPF_{s_2} = -\frac{s_2 - ref}{s_2} = -0.3125$$

Since the features s_3 and s_4 are within the range, they are calculated regularly as before. Thus,

$$s_3 = |X + \frac{ref - s_3}{s_3' + (BV * X)}| = 0.915$$

Similarly,

$$s_4 = |X - \frac{s_4 - ref}{s_4}| = 1.083$$

Notice that s_2 receives a lower penalty than s_1. Thus, s_2 in this case ranks higher than s_1 as it is semantically better. Also, s_4 receives a higher score than s_3.

5.8 Algorithm Pseudo-code

In this section, we present a Pseudo-code for the complete X-Algorithm. It incorporates all the options and modes discussed in this Chapter.

Algorithm 4: The X-Algorithm

```
Data: Services in the Registry
Result: Services ranked based on users' inputs
Input: Query, Modes(Exact,Best), RangeMode(RP), Essential(ESS), ShowBest(SABF),
       RegularWeight(RW),SystemAccuracy(acc)
begin
    Initialization
    for $i \leftarrow NumberOfServices$ do
        for $j \leftarrow NumberOfQueryFeature$ do
            $X \leftarrow CheckFeatureSemantic(j)$  // X =-1(MB),+1(LB),0(EB)
            $Mode \leftarrow CheckFeatureMode(j)$  // MODE=1(ExactMode), 0(BestMode)
            if $q_i = 0$ then
                $PM_{ij} \leftarrow 0$
            else if $s_{ij} = 0$ OR $s_{ij} = -1$ then
                $PM_{ij} \leftarrow -2$
            else if $X = 0$ then
                $PM_{ij} \leftarrow EBHandler()$
            else if $s_{ij} = q_i$ then
                $PM_{ij} \leftarrow 1$
            else if $s_{ij} > q_i$ then
                $if(Mode = 1)\ X \leftarrow +1$
                $PM_{ij} \leftarrow scale(|X - \frac{s_{ij}-q_i}{s_{ij}}|)$
                if RP=1 AND $s_{ij} > q_{iupper}$ then  // when Range= ON
                    $ref = (upperValue + lowerValue)/2$
                    $PM_{ij} = -scale(\frac{ref - s_{ij}}{s_{ij}})$
                end if
            else
                $BV = X * acc$  // BoostValue
                $if(Mode == 1)\ X \leftarrow -1$  // this only works if EF=1
                $s'_{ij} \leftarrow q_i + (q_i - s_{ij})$
                $PM_{ij} \leftarrow scale(|X + \frac{s'_{ij}-q_i}{s'_{ij}+BV}|)$
                if RP=1 AND $s_{ij} < q_{ilower}$ then  // when Range=ON
                    $ref = (upperValue + lowerValue)/2$
                    $PM_{ij} = -scale(\frac{ref - s'_{ij}}{s'_{ij}+BV})$
                end if
            end if
            if $PM_{ij} \geq 1$ then  // when AllBest option is ON
                $AllBest_{ij} \leftarrow 1$
            else
                $AllBest_{ij} \leftarrow PM_{ij}$
            end if
        end for
    end for
    $W_E = W_E * \frac{W_R}{1000}$  // when Essential option is ON
    $W = W_E + W_R$
    $FinalRank = W * PM$
    for $i \leftarrow 0$ to $size(AllBest)$ do
        if $SumColumn(AllBest_i) \leq NumberOfQueryFeature$ then
            $FR(i) \leftarrow FR(i) + SABFV$
        end if
    end for
end
```

5.9 Common Problems and Solutions

In this section we highlight the significance and originality of the X-Algorithm. We comment on the common problems associated with vector-based algorithms, and highlight the methods in X-Algorithm that overcome these problems.

5.9.1 Missing Feature Problem

This problem occurs when vectors are not of the same length which is claimed to be a limitation in vector-based algorithms [WMZ10] [RB09]. That is, when a feature specified in a *Query* is missing in one or more *Service*(s). Basically, this results in uneven matrix which cannot be rigorously treated with vector algebra. In order to overcome this obstacle, we assign a $penalty$ of -2, which is the maximum penalty ($maxPenalty$) produced by the algorithm, to each feature missing in a *Service*. This will make the matrix even and valid for vector calculations. At the same time it will make the missing feature receive lowest score since it does not offer what the user has requested. Then, at the Multiplication Phase, the penalty affects the total rank based on the weight assigned to that missing feature in *Query*.

5.9.2 Mathematical Problems

Some ranking algorithms, based on *Similarity Measure*, are prone to mathematical problems such as division by zero and square root of minus values. Our ranking algorithm is based on the *Similarity Measure* SM_X is not prone to this problem because of two reasons. First, because we adopted symmetry and our denominator is always the image around the value of *Query*. Consider the fraction

$$\frac{|q-s|}{y}$$

in SM_X, where $y = s$ if $s > q$ and $y = s' = |q-s| + q$ if $s < q$. If $s = 0$, and because $q > 0$ it is the case that $s < q$. Thus, the denominator is $y = s'$ which makes the denominator greater than 0. In case $q = 0$ and $s \neq 0$, it is the case $s > q$ and thus the denominator is not 0. Division by zero is likely only when both q and s are 0. However in this case $s = q$, which we examine before we calculate similarity.

Currently, we consider the values 0 in *Query* as an undesired feature, while 0 in *Service* features as a missing feature. Nevertheless, our algorithm is capable of dealing with it naturally. However, we have excluded those cases since we are aiming for employing our algorithm in online *Services* trading where the value 0 is inapplicable as a feature.

5.9.3 Compensation and Non-compensation Problem

Although we have come across this feature in brief in Section 5.6, we repeat it here for its importance and because it is a common concern. As explained earlier, compensatory algorithms allow the good features of a *Service* to cover for the bad features, while non-compensatory algorithms disallow that. Because we are using an additive aggregation method, where we sum all scores of all features of a *Service* to produce the *Service*'s final rank, we are prone to this problem. However, since we claim we want to provide a real-life-like ranking algorithm, we should allow compensation as it is present even in real life. Nevertheless, we should also minimize it upon request. Therefore, we have provided the Option, $AllBest$ to minimize this problem. Briefly, the problem is minimized by keeping all $rewards$ to minimum reward value. This will ensure that the affect of the good features are degraded to minimal.

5.10 Summary

In this chapter, we have given a detailed development of X-Algorithm. We have started by defining the concepts of penalty and reward that we have adopted in our *Similarity Measure*. Then, we have explained each option of the algorithm in terms of the option's goal, and how it is integrated into our algorithm. With each option, we have provided an inclusive example that covers all the cases of the option. After that, we have introduced a pseudo code that includes all algorithm options and functions. Finally, we commented on the most common problems of ranking algorithms and explained how X-Algorithm resolves them.

Chapter 6

Accuracy, Complexity, and Performance

In this chapter we discuss the accuracy of the X-Algorithm by running the algorithm on a selected subset of Google Data Store and manually validating the results. A theoretical analysis of the algorithm complexity is also discussed. Finally, we discuss the runtime performance of the X-Algorithm on two sets of randomly selected *Service* data.

6.1 Accuracy

We provide a case study to manually examine accuracy. In this case study, we apply our algorithm on a real-world data. This data is Google Application Store's data [Inc08]. First, we are going to explain why we chose this data. Then, we will introduce the problem we are trying to solve through this case study and propose our solution. After that, we collect and analyse the Google Store's data. Finally, we apply our solution on the data and show our findings.

6.1.1 Why Google's data?

The reason why we chose Google Store's data in particular is behind the nature of its data. Since it has different semantics, i.e. *MB*, *LB* and *EB*, it allows us to examine our algorithm and the potential situations it unfolds. Also, each application or product in Google Store is multi-featured. This also complies with our algorithm as we make it to allow users to define multi-features in *Query*. Last, Google Store contains huge number of data that is commonly queried. Showing that we could improve on the current solution for such data proves our algorithm's practicality and significance.

6.1.2 Problem of current ranking algorithm in Google Store

Currently, Google Store ranking is based on implicit criteria ranking. These criteria such as reviews, number of downloads, and level of usage are used to rank applications. However, users of Google Store don't have the ability to *Query* based on a specific request. For example, if someone wants to look for cheapest application in a certain category, he/she cannot do so in the current Google solution. Similarly, users can't look for cheapest application with highest rating. This limits users to see only what the implicit ranking algorithm has to offer.

To overcome this limitation, we propose the X-Algorithm as a solution. By applying our algorithm on top of the current ranking algorithm, we can allow users to manipulate the results list based on their specific requirements. These requirements can contain different data types and different modes as we will explain in next sections. Providing the different modes, semantics and options, as in X-Algorithm, we provide the customers to compose a wider range of Queries and get a more accurate ranking for their Queries.

6.1.3 Collecting and Analysing Google Store's data

To be able to apply our algorithm on any data, we have to understand the data in terms of (1) the number of attributes, and (2) potential data types. The data of Google contains four attributes. These attributes are, name of application, rating, price and a flag that indicates whether application is free or not. The types of data are as follows:

- Name of application: The type of the name is string in all cases. Thus, it is EB attribute.
- Rating: The rating is better as it increases. This means it is MB attribute.
- Price: The price is better as it decreases. Hence, it is considered as LB attribute.
- Free application flag: It is a boolean data type which means it is EB attribute.

We also include two other data, but we called it *unrankable data* as they are not included as a criteria in ranking. These are

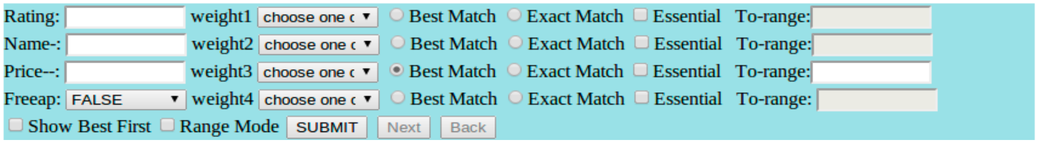
- Link for each product: This is necessary to make sure that we have mapped the data correctly.
- Installed flag: This indicates if a user has installed the application already. We have included it to be able to involve it in the ranking criteria in case we want it later on.

Having defined each attribute in Google's data, we need now to define a method to collect this data. Since this data is accessible through a website, we need to extract the data we need to apply our algorithm.

Remark. *This extraction is for scientific reasons only. We do not perform, by any means, republishing (Web Scraping) or reusing Google's Data publicly.*

The stages to extract Google Store's data are explained below.

1. Parse the html file: We coded a small script that reads the html file and parses it.



Rating:		weight1	choose one c ▾	○ Best Match	○ Exact Match	☐ Essential	To-range:
Name-:		weight2	choose one c ▾	○ Best Match	○ Exact Match	☐ Essential	To-range:
Price--:		weight3	choose one c ▾	● Best Match	○ Exact Match	☐ Essential	To-range:
Freeap:	FALSE ▾	weight4	choose one c ▾	○ Best Match	○ Exact Match	☐ Essential	To-range:

Figure 26: User Interface to build Query

2. Write the parsed version of html to a file: We have made the script to write each html to a different file. This is to improve the testability and fixing data errors if any.

3. Read all files and write it to one large file. Thus, the algorithm can be applied on a single file.

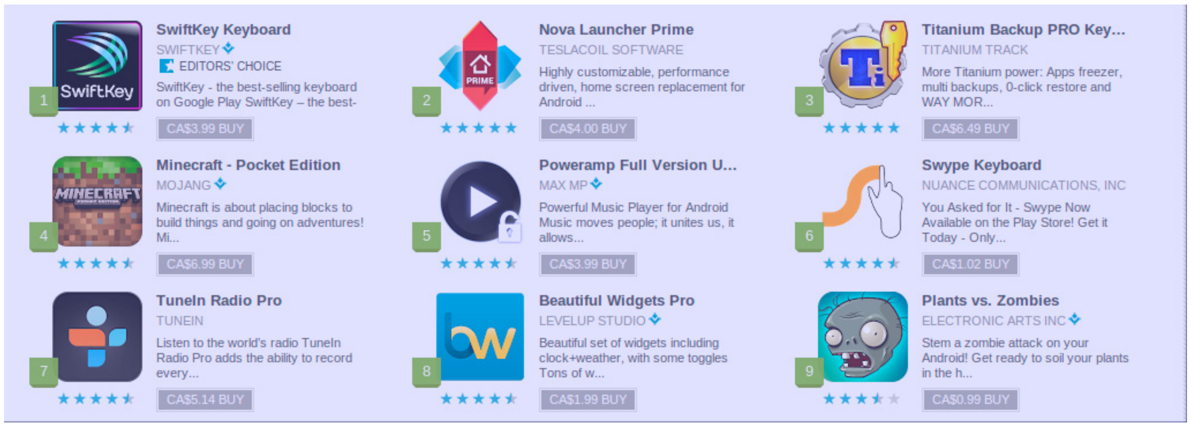
4. Apply the algorithm on the large file.

6.1.4 Applying our solution

In order to make our solution more practical, we have implemented our algorithm as an HTTP server. This server gets the request of the user from the browser, ranks the results based on the request, and sends the ranked results back to the user. Thus, we needed to implement an interface to the user through which requests can be made. In the user interface shown in Figure 26, we have included all options allowed by the algorithm.

For the sake of simplicity, we have included only nine Services in the file to be ranked by our algorithm. This is to show how we can manipulate the top nine data based on users requirement. At the time of testing, the top nine Services in Google Store are the ones shown in Table 23 and Figure 27. Also, we considered the variables $EssentialAccuracy$ and *AlgorithmAccuracy*, to be pre-set to the values 0.01, 0.1 respectively. We also assumed the semantics of the $Price$ feature is LB, the semantics of $Rating$ feature is MB, and the semantics of both $Name$ and $isFree$ features is EB.

In the following paragraphs, we define a number of user Queries, and explain the differences between the rankings produced by the X-Algorithm and the original Google rankings.


1 SwiftKey Keyboard
SWIFTKEY
EDITORS' CHOICE
SwiftKey - the best-selling keyboard on Google Play SwiftKey – the best-
CA$3.99 BUY

2 Nova Launcher Prime
TESLACOIL SOFTWARE
Highly customizable, performance driven, home screen replacement for Android ...
CA$4.00 BUY

3 Titanium Backup PRO Key...
TITANIUM TRACK
More Titanium power: Apps freezer, multi backups, 0-click restore and WAY MOR...
CA$6.49 BUY

4 Minecraft - Pocket Edition
MOJANG
Minecraft is about placing blocks to build things and going on adventures! Mi...
CA$6.99 BUY

5 Poweramp Full Version U...
MAX MP
Powerful Music Player for Android Music moves people; it unites us, it allows...
CA$3.99 BUY

6 Swype Keyboard
NUANCE COMMUNICATIONS, INC
You Asked for It - Swype Now Available on the Play Store! Get it Today - Only...
CA$1.02 BUY

7 TuneIn Radio Pro
TUNEIN
Listen to the world's radio TuneIn Radio Pro adds the ability to record every...
CA$5.14 BUY

8 Beautiful Widgets Pro
LEVELUP STUDIO
Beautiful set of widgets including clock+weather, with some toggles Tons of w...
CA$1.99 BUY

9 Plants vs. Zombies
ELECTRONIC ARTS INC
Stem a zombie attack on your Android! Get ready to soil your plants in the h...
CA$0.99 BUY

Figure 27: A snap shot of the top nine Google Data as shown in the website at the time of writing

Rank	Name	Price	Rating	isFree
1	Swift Keyboard	3.9	4.6	False
2	Nova Launcher Prime	4.0	4.8	False
3	Titanium Backup Pro Key root	6.4	4.8	False
4	Minecraft pocket edition	6.9	4.5	False
5	Poweramp Full version Unlocker	3.9	4.7	False
6	Swype keyboard	1.0	4.5	False
7	TuneIn Radio Pro	5.1	4.5	False
8	Beautiful Widgets Pro	1.9	4.3	False
9	Plants VS Zombies	0.9	3.4	False

Table 23: The top nine Services at the time of testing on Google Play Store

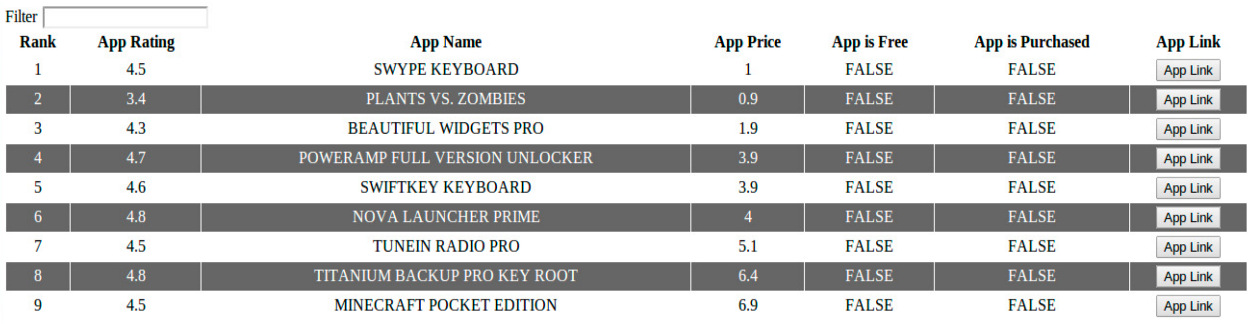
Potential Queries

1. *Regular weighted Query which includes both matching modes*

   - **Case1:** *When the mode of Price feature is set to Exact and the mode of Rating is set to Best, the Query shown in Table 24 is applied to Google Data Store.*

	Name	Price	Rating	isFree
VALUES	plants	1	2	true
WEIGHTS	low	critical	normal	low
MODE	-	E	B	-
ESS	0	0	0	0
RANGE	0			
SABF	0			

Table 24: Query for Case1


Filter

Rank	App Rating	App Name	App Price	App is Free	App is Purchased	App Link
1	4.5	SWYPE KEYBOARD	1	FALSE	FALSE	App Link
2	3.4	PLANTS VS. ZOMBIES	0.9	FALSE	FALSE	App Link
3	4.3	BEAUTIFUL WIDGETS PRO	1.9	FALSE	FALSE	App Link
4	4.7	POWERAMP FULL VERSION UNLOCKER	3.9	FALSE	FALSE	App Link
5	4.6	SWIFTKEY KEYBOARD	3.9	FALSE	FALSE	App Link
6	4.8	NOVA LAUNCHER PRIME	4	FALSE	FALSE	App Link
7	4.5	TUNEIN RADIO PRO	5.1	FALSE	FALSE	App Link
8	4.8	TITANIUM BACKUP PRO KEY ROOT	6.4	FALSE	FALSE	App Link
9	4.5	MINECRAFT POCKET EDITION	6.9	FALSE	FALSE	App Link

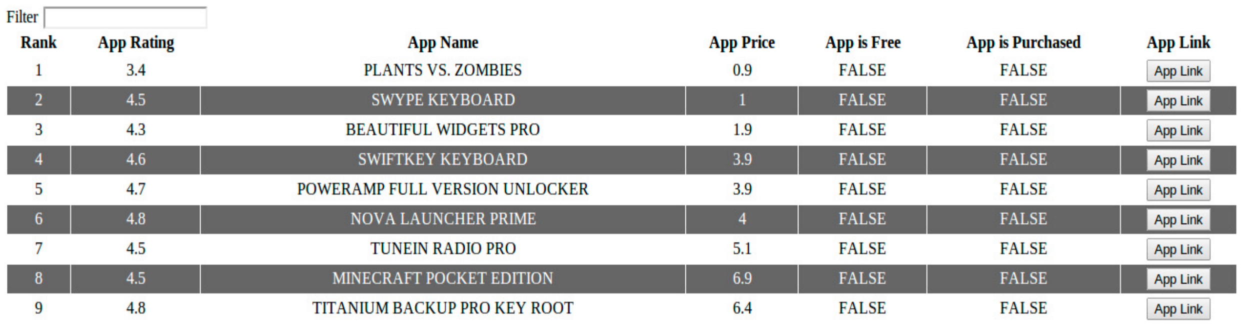
Figure 28: Results for Case1

- **Case2:** *When the mode of Price feature is set to Exact and the mode of Rating is set to Best, the Query shown in Table 25 is applied to Google Data Store.*

	Name	Price	Rating	isFree
VALUES	plants	1	2	true
WEIGHTS	low	critical	normal	low
MODE	-	B	E	-
ESS	0	0	0	0
RANGE	0			
SABF	0			

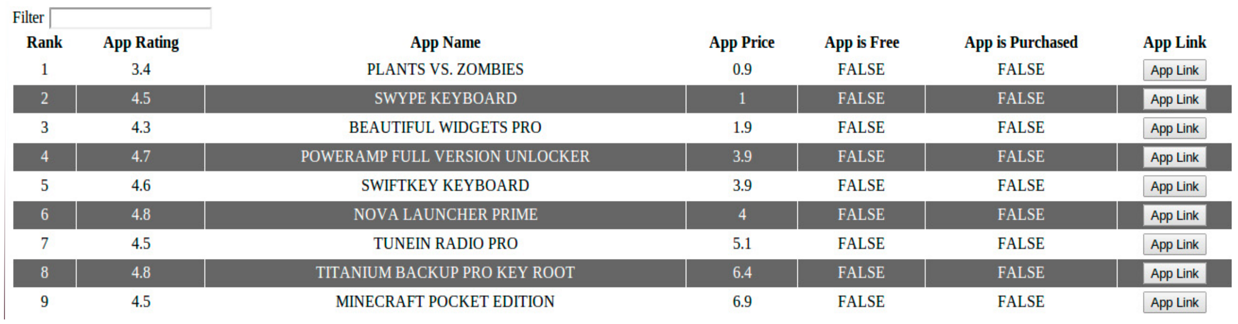
Table 25: Query for Case2

Notice that in this mode the second Service in Figure 28 becomes the first Service in Figure 29.


Filter

Rank	App Rating	App Name	App Price	App is Free	App is Purchased	App Link
1	3.4	PLANTS VS. ZOMBIES	0.9	FALSE	FALSE	App Link
2	4.5	SWYPE KEYBOARD	1	FALSE	FALSE	App Link
3	4.3	BEAUTIFUL WIDGETS PRO	1.9	FALSE	FALSE	App Link
4	4.6	SWIFTKEY KEYBOARD	3.9	FALSE	FALSE	App Link
5	4.7	POWERAMP FULL VERSION UNLOCKER	3.9	FALSE	FALSE	App Link
6	4.8	NOVA LAUNCHER PRIME	4	FALSE	FALSE	App Link
7	4.5	TUNEIN RADIO PRO	5.1	FALSE	FALSE	App Link
8	4.5	MINECRAFT POCKET EDITION	6.9	FALSE	FALSE	App Link
9	4.8	TITANIUM BACKUP PRO KEY ROOT	6.4	FALSE	FALSE	App Link

Figure 29: Results for Case2


Filter

Rank	App Rating	App Name	App Price	App is Free	App is Purchased	App Link
1	3.4	PLANTS VS. ZOMBIES	0.9	FALSE	FALSE	App Link
2	4.5	SWYPE KEYBOARD	1	FALSE	FALSE	App Link
3	4.3	BEAUTIFUL WIDGETS PRO	1.9	FALSE	FALSE	App Link
4	4.7	POWERAMP FULL VERSION UNLOCKER	3.9	FALSE	FALSE	App Link
5	4.6	SWIFTKEY KEYBOARD	3.9	FALSE	FALSE	App Link
6	4.8	NOVA LAUNCHER PRIME	4	FALSE	FALSE	App Link
7	4.5	TUNEIN RADIO PRO	5.1	FALSE	FALSE	App Link
8	4.8	TITANIUM BACKUP PRO KEY ROOT	6.4	FALSE	FALSE	App Link
9	4.5	MINECRAFT POCKET EDITION	6.9	FALSE	FALSE	App Link

Figure 30: Results for Case3

2. *Best flag is ON*

   Case3: *We consider the same Query shown in Table 24 with AllBest Option set to "ON"*

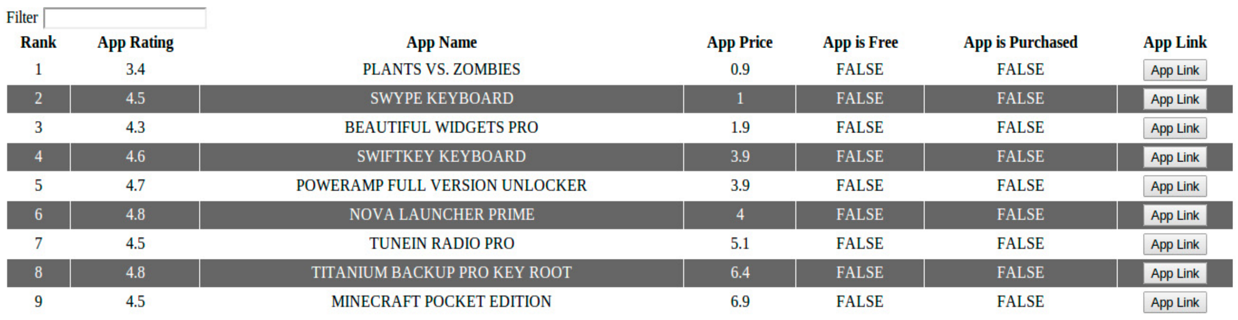
   Notice the change from the results of Case1. This is because the second Service in Figure 30 is closer to the user request than the first Service. That is, since there is a partial match in the name, and a compliance with the requested rating, the only missing part for a perfect match is 0.1 in price. Thus, when AllBest Option is "ON" the algorithm ranks the Service which was ranked second in Figure 28 to the top position in Figure 30.

3. *Essential flag is ON*

   - **Case4:** *When Price is set to Essential attribute and Best Mode is "ON", the Query shown in Table 26 is applied to Google Data Store.*

	Name	Price	Rating	isFree
VALUES	swiftkey	1	2	true
WEIGHTS	low	critical	normal	low
MODE	-	B	B	-
ESS	0	1	0	0
RANGE	0			
SABF	0			

Table 26: Query for Case4


Filter

Rank	App Rating	App Name	App Price	App is Free	App is Purchased	App Link
1	3.4	PLANTS VS. ZOMBIES	0.9	FALSE	FALSE	App Link
2	4.5	SWYPE KEYBOARD	1	FALSE	FALSE	App Link
3	4.3	BEAUTIFUL WIDGETS PRO	1.9	FALSE	FALSE	App Link
4	4.6	SWIFTKEY KEYBOARD	3.9	FALSE	FALSE	App Link
5	4.7	POWERAMP FULL VERSION UNLOCKER	3.9	FALSE	FALSE	App Link
6	4.8	NOVA LAUNCHER PRIME	4	FALSE	FALSE	App Link
7	4.5	TUNEIN RADIO PRO	5.1	FALSE	FALSE	App Link
8	4.8	TITANIUM BACKUP PRO KEY ROOT	6.4	FALSE	FALSE	App Link
9	4.5	MINECRAFT POCKET EDITION	6.9	FALSE	FALSE	App Link

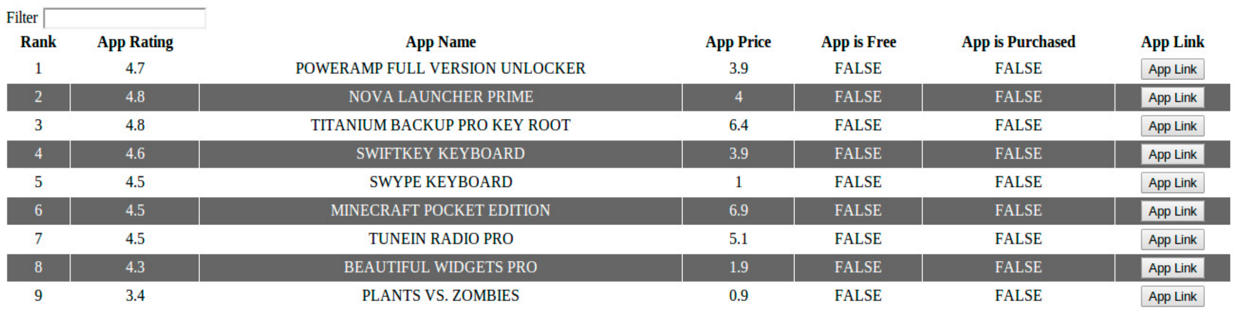
Figure 31: Results for Case4

In Figure 31 the Services ranked fourth and fifth have the same Price. However, the algorithm has ranked the item with application name "Swiftkey" higher than the other one, because of the name criterion. This illustrates that although Essential Option is "ON" ranking maybe affected with respect to other non-essential attributes.

- **Case5:** *When Rating attribute is set to Essential and mode is Exact, the Query defined in Table 27 is applied to Google Data Store.*

	Name	Price	Rating	isFree
VALUES	swiftkey	1	4.7	true
WEIGHTS	low	critical	normal	low
MODE	-	B	E	-
ESS	0	0	1	0
RANGE	0			
SABF	0			

Table 27: Query for Case5


Filter

Rank	App Rating	App Name	App Price	App is Free	App is Purchased	App Link
1	4.7	POWERAMP FULL VERSION UNLOCKER	3.9	FALSE	FALSE	App Link
2	4.8	NOVA LAUNCHER PRIME	4	FALSE	FALSE	App Link
3	4.8	TITANIUM BACKUP PRO KEY ROOT	6.4	FALSE	FALSE	App Link
4	4.6	SWIFTKEY KEYBOARD	3.9	FALSE	FALSE	App Link
5	4.5	SWYPE KEYBOARD	1	FALSE	FALSE	App Link
6	4.5	MINECRAFT POCKET EDITION	6.9	FALSE	FALSE	App Link
7	4.5	TUNEIN RADIO PRO	5.1	FALSE	FALSE	App Link
8	4.3	BEAUTIFUL WIDGETS PRO	1.9	FALSE	FALSE	App Link
9	3.4	PLANTS VS. ZOMBIES	0.9	FALSE	FALSE	App Link

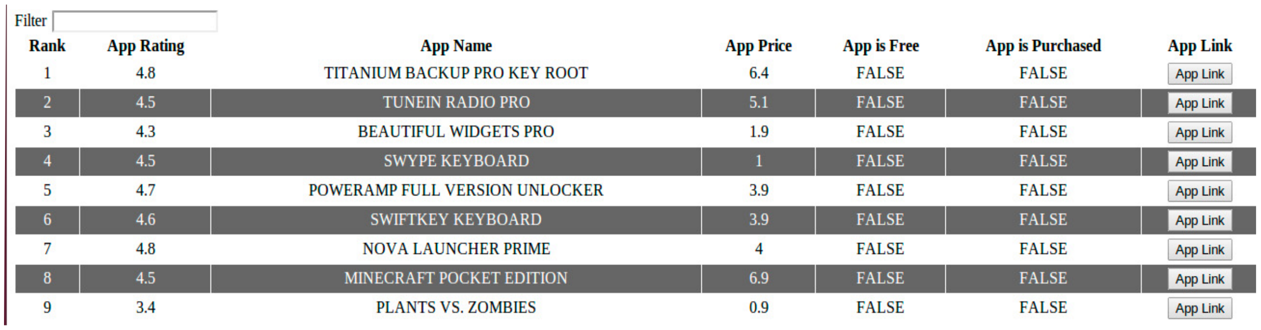
Figure 32: Results for Case5

Since Rating attribute is set to Essential in Exact Mode, the item with application name "POWERAMP FULL VERSION UNLOCKER" and Rating 4.7 is ranked the highest in Figure 32. There are two Services with Rating 4.8 and one Service with Rating 4.6 and all of them are at the same distance from 4.7. However, the algorithm prefers to rank the two Services with Rating 4.8 higher than the Service with Rating 4.6, because the semantics for Rating attribute is MB.

- **Case6:** *When Name attribute is set to Essential, the Query shown in Table 28 is applied to Google Data Store.*

	Name	Price	Rating	isFree
VALUES	pro	4	1	true
WEIGHTS	critical	low	critical	low
MODE	-	B	B	-
ESS	1	0	0	0
RANGE	0			
SABF	0			

Table 28: Query for Case6



Filter

Rank	App Rating	App Name	App Price	App is Free	App is Purchased	App Link
1	4.8	TITANIUM BACKUP PRO KEY ROOT	6.4	FALSE	FALSE	App Link
2	4.5	TUNEIN RADIO PRO	5.1	FALSE	FALSE	App Link
3	4.3	BEAUTIFUL WIDGETS PRO	1.9	FALSE	FALSE	App Link
4	4.5	SWYPE KEYBOARD	1	FALSE	FALSE	App Link
5	4.7	POWERAMP FULL VERSION UNLOCKER	3.9	FALSE	FALSE	App Link
6	4.6	SWIFTKEY KEYBOARD	3.9	FALSE	FALSE	App Link
7	4.8	NOVA LAUNCHER PRIME	4	FALSE	FALSE	App Link
8	4.5	MINECRAFT POCKET EDITION	6.9	FALSE	FALSE	App Link
9	3.4	PLANTS VS. ZOMBIES	0.9	FALSE	FALSE	App Link

Figure 33: Results for Case6

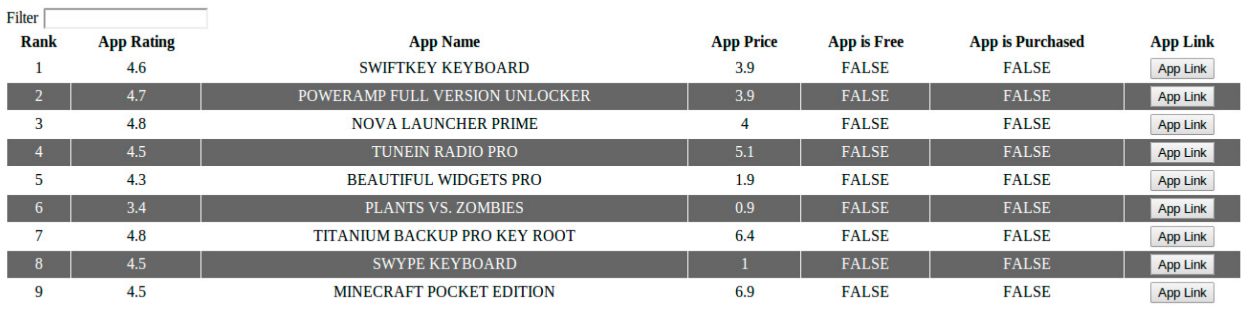
If we look at the first three Services in Figure 33, we find all of them contain the word "PRO" which is the one used in the Query. However, the algorithm preferred the one on top over the others that contain "PRO", because of the value of Rating attribute. The value of Rating attribute is considered because its weight in Query has been set to critical and the weight of Price has been set to low. Thus, when essential requirement is met which is the existence of the word "PRO", regular ranking for other non-essential properties is performed.

4. *Range flag is ON (only works with Best Mode)*

- **Case7:** *When the Range Option is "ON" with Best Mode for Price feature, the Query in Table 29 is applied to Google Data Store.*

	Name	Price	Rating	isFree
VALUES	Nova	5.1-3	1	true
WEIGHTS	low	critical	normal	low
MODE	-	B	B	-
ESS	0	0	0	0
RANGE	1			
SABF	0			

Table 29: Query for Case7



Filter

Rank	App Rating	App Name	App Price	App is Free	App is Purchased	App Link
1	4.6	SWIFTKEY KEYBOARD	3.9	FALSE	FALSE	App Link
2	4.7	POWERAMP FULL VERSION UNLOCKER	3.9	FALSE	FALSE	App Link
3	4.8	NOVA LAUNCHER PRIME	4	FALSE	FALSE	App Link
4	4.5	TUNEIN RADIO PRO	5.1	FALSE	FALSE	App Link
5	4.3	BEAUTIFUL WIDGETS PRO	1.9	FALSE	FALSE	App Link
6	3.4	PLANTS VS. ZOMBIES	0.9	FALSE	FALSE	App Link
7	4.8	TITANIUM BACKUP PRO KEY ROOT	6.4	FALSE	FALSE	App Link
8	4.5	SWYPE KEYBOARD	1	FALSE	FALSE	App Link
9	4.5	MINECRAFT POCKET EDITION	6.9	FALSE	FALSE	App Link

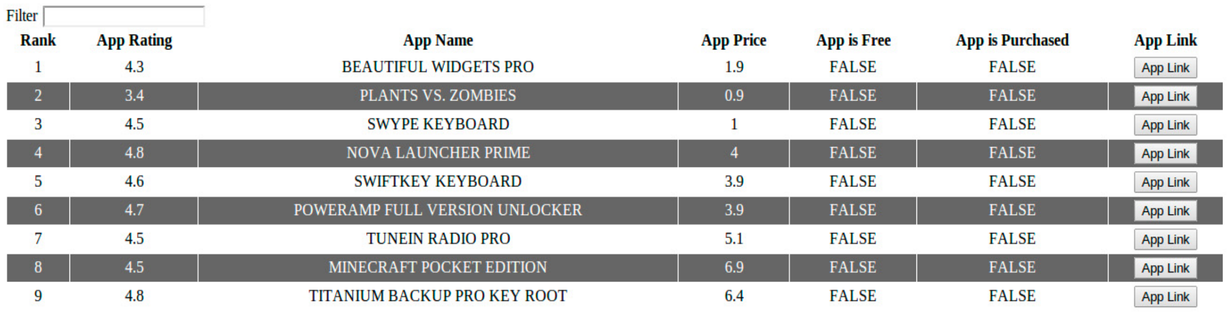
Figure 34: Results for Case7

In Figure 34, notice how the semantics for Ranges play a pivotal role in ranking. That is because the semantic for Price is LB. That is, lowest values in the range are better than higher values. Also, notice how values on top are within the range. This shows the importance given to the feature with Range Option compared to other features. The reason is that the attribute values that don't comply with the set range are punished, which affect their scores and thus affect their final ranks.

- **Case8:** *When Range Option is "ON" with Best Mode for Rating attribute, the Query in Table 30 is applied on Google Data Store.*

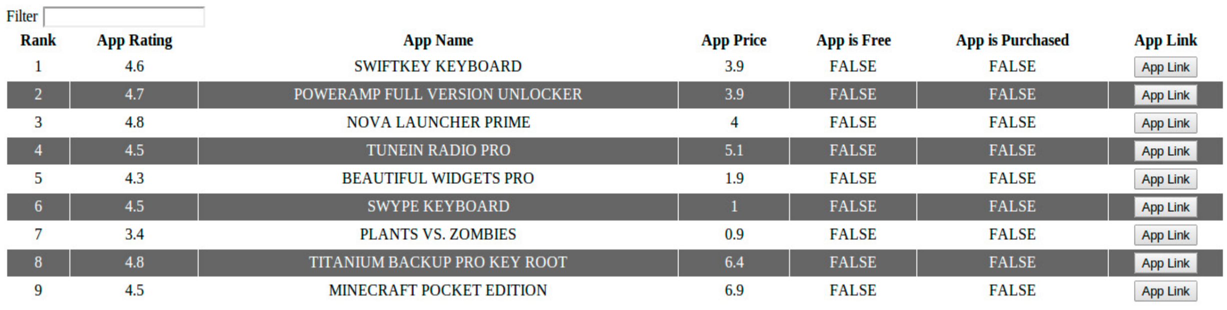
	Name	Price	Rating	isFree
VALUES	Nova		4.3-3.4	true
WEIGHTS	low	normal	critical	low
MODE	-	B	B	-
ESS	0	0	0	0
RANGE	1			
SABF	0			

Table 30: Query for Case8


Filter

Rank	App Rating	App Name	App Price	App is Free	App is Purchased	App Link
1	4.3	BEAUTIFUL WIDGETS PRO	1.9	FALSE	FALSE	App Link
2	3.4	PLANTS VS. ZOMBIES	0.9	FALSE	FALSE	App Link
3	4.5	SWYPE KEYBOARD	1	FALSE	FALSE	App Link
4	4.8	NOVA LAUNCHER PRIME	4	FALSE	FALSE	App Link
5	4.6	SWIFTKEY KEYBOARD	3.9	FALSE	FALSE	App Link
6	4.7	POWERAMP FULL VERSION UNLOCKER	3.9	FALSE	FALSE	App Link
7	4.5	TUNEIN RADIO PRO	5.1	FALSE	FALSE	App Link
8	4.5	MINECRAFT POCKET EDITION	6.9	FALSE	FALSE	App Link
9	4.8	TITANIUM BACKUP PRO KEY ROOT	6.4	FALSE	FALSE	App Link

Figure 35: Results for Case8


Filter

Rank	App Rating	App Name	App Price	App is Free	App is Purchased	App Link
1	4.6	SWIFTKEY KEYBOARD	3.9	FALSE	FALSE	App Link
2	4.7	POWERAMP FULL VERSION UNLOCKER	3.9	FALSE	FALSE	App Link
3	4.8	NOVA LAUNCHER PRIME	4	FALSE	FALSE	App Link
4	4.5	TUNEIN RADIO PRO	5.1	FALSE	FALSE	App Link
5	4.3	BEAUTIFUL WIDGETS PRO	1.9	FALSE	FALSE	App Link
6	4.5	SWYPE KEYBOARD	1	FALSE	FALSE	App Link
7	3.4	PLANTS VS. ZOMBIES	0.9	FALSE	FALSE	App Link
8	4.8	TITANIUM BACKUP PRO KEY ROOT	6.4	FALSE	FALSE	App Link
9	4.5	MINECRAFT POCKET EDITION	6.9	FALSE	FALSE	App Link

Figure 36: Results for Case9

Unlike Case7, higher values within the range are ranked higher because the semantics of the Rating attribute is MB.

- **Case9:***When Essential and Range Options with Best Mode are set for Price, the Query in Table 31 is applied to Google Data Store.*

	Name	Price	Rating	isFree
VALUES	Nova	5.1-2	1	true
WEIGHTS	low	critical	normal	low
MODE	-	B	B	-
ESS	0	1	0	0
RANGE	1			
SABF	0			

Table 31: Query for Case9

In Figure 36, the Services are ranked based on the Price attribute as it is set to Essential. Thus, values that match the minimum values within the range are ranked higher as the semantic of Price is LB. However, values outside the range are ranked based on its closeness to the range. Similar to Essential Option, it is based on the LB semantic as well. That is, if two values differ from the range by the same distance on different directions, the values in favour of the semantics ranks higher than the other.

6.2 Complexity

In the pseudo code, we have tried to minimize the number of loops to be executed. No nested loops exist in the code except for the two main ones, one for the rows and the other for columns of the matrix PM. Within these two loops the modes and semantic preferences, and other options are included. The only thing that remains outside the loop is the final sorting of the total ranks. Thus, the algorithmic complexity of X-Algorithm is twofold; (1) the time necessary to calculate the aggregated ranks, and (2) the time necessary to sort these aggregated ranks. Since the first one does not have any nested loop, its complexity is $O(N_f * N_s)$ where N_f is the number of features in each *Service*, and N_s is the number of *Services* to be ranked. Quicksort algorithm is used for the final ordering of *Services*. Hence, the total complexity is,

$$Time_{X\_Algo} = T(N_f, N_s) = O(N_s * N_f) + O(N_s log(N_s))$$

It is reasonable to assume that the number of features N_f is fixed for an application. That is, N_f is a constant. Hence, asymptotically the complexity is $O(N_s log(N_s))$, where N_s is the number of *Services* in the final order.

6.3 Performance

We evaluate the performance of X-Algorithm by measuring the execution times of the algorithm on randomly generated large datasets. We generated two different types of datasets. In one type of dataset we fixed the number of features and varied the number of *Services* to produce different datasets with asymptotically increasing size. In the second type of dataset we fixed the number of *Services* and varied the number of features to produce different instances of this type.

6.3.1 Type 1 Dataset: Fixed Number of Features and Variable number of Services

We created ten datasets. The first dataset has 10000 services. Successively we increased the size by 10000 to create the rest of the datasets. We fixed the number of features to be six with semantics defined below.

$$Features : [LB, LB, MB, MB, EB, EB]$$

where LB,MB and EB are the different semantics for the features.

Fixing the number of features is important to measure the effect of changing the number of *Services* on the execution time. Also, for a fair estimation we have included all semantics in each *Service*. We ran the X-Algorithm for each dataset three times and calculated the average of the execution times. These results are listed in Table 32. The results show a linear runtime behaviour, as shown in Figure(A) 37.

Group	# of Alternatives	Execution Time (ms)
1	10,000	137.964
2	20,000	212.189
3	30,000	291.361
4	40,000	349.656
5	50,000	414.279
6	60,000	473.738
7	70,000	553.305
8	80,000	624.474
9	90,000	666.083
10	100,000	728.330

Table 32: Execution time when number of *Services* increases

As the figure shows, the linear model shows a good fit on the results. This is also shown in

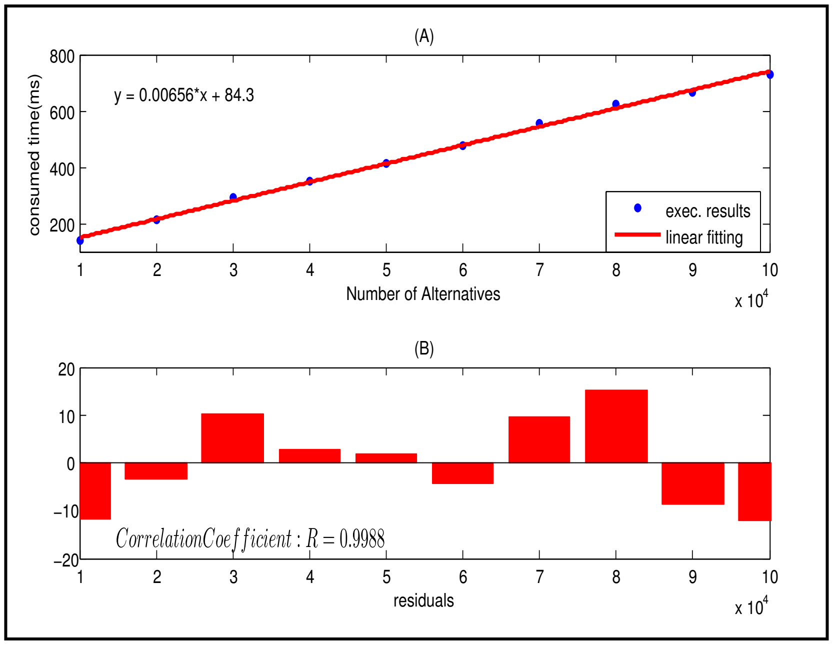


Figure 37: This figure shows how the results exhibits a linear growth

Figure(B) 37, where the correlation coefficient (R) was calculated. This function is a statistical function that is used to measure the goodness of the fitting model. It produces a number between $(1, -1)$, where $+1$ shows the best fit and -1 indicates the worst. As the figure shows, R is 0.9988 which is considered to be a good fit.

6.3.2 Type2 Dataset: Variable number of Features and Fixed Number of Services

Here we examine the performance of the X-Algorithm when the number of *Services* is fixed to 1000, and the number of features is varied. Because our algorithm is a user-centric, the number of features cannot be too high. That is, if features are too many, *Query* becomes hard to be built by the user. However, it is important to capture the growth of the execution time when features increase. We ran the algorithm three times on each dataset. The first data set has ten attributes and successively the number of attributes is increased by 10 to produce nine other datasets. For all datasets, the *Services* are fixed. In all runs, we considered all the three semantics. In Table 33 we list the average execution times for the ten datasets.

Group	# of Features	Execution Time (ms)
1	10	179.54
2	20	290.969
3	30	419.169
4	40	566.175
5	50	726.794
6	60	802.669
7	70	960.520
8	80	1096
9	90	1245
10	100	1373.667

Table 33: Execution time evaluation when number of features increases.

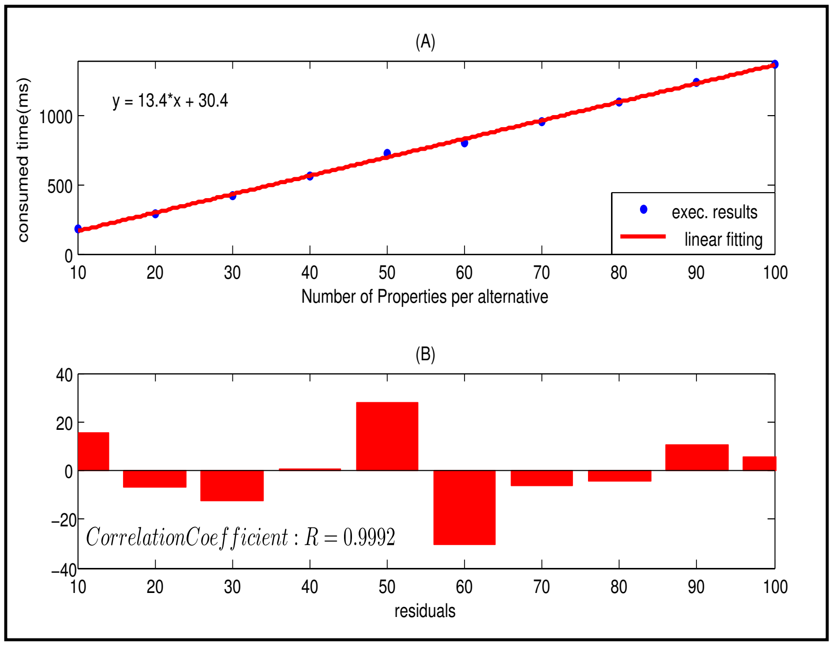


Figure 38: This figure shows how the results exhibits a linear growth

The results show that the runtime behaviour is once again linear. We calculated the Correlation Coefficient R to measure the validity of the linear model. The Correlation Coefficient shows that the linear model is a good fit. Figure(A) 38 and Figure(B) 38 show the results of this experiment.

6.4 Summary

In this chapter we have provided an overall evaluation for the X-Algorithm from three different perspectives; Results Accuracy, Algorithmic Complexity, and Runtime Performance. In our evaluation, we exposed our algorithm to a real-world data, and randomly selected high volume data to see examine its robustness and practicality. We found that the X-Algorithm have performed exceptionally in all settings. This shows the algorithm significance and ability to be used in practice for wide range of applications.

Chapter 7

Ranking Composite Services

A *Simple Service* is *atomic*, in the sense that it cannot be split into smaller *Services*. A *Complex Service* is obtained by putting together one or more Simple *Services*. It is also possible to combine a simple *Service* with a complex *Service* to produce another complex *Service*. Since users are to construct requests for complex *Services*, we assume that a complex *Service* is a package of simple *Services*. In *Service* oriented computing researchers have studied different methods, called *compositions*, for putting together *Services* as a complex *Service* [Ibr12]. In this thesis, we use the terms complex *Service* and composite *Service* interchangeably.

In a social setting composite *Services* arise and are in great demand. As an example, a travel package offered by a *Service* provider is a complex *Service*, which typically includes air travel, hotel accommodations, car rental and other simple *Services*. A consumer can buy the whole package offered by a *Service* provider, however the consumer cannot buy an individual *Service* within the complex *Service*. This means that, in general, vacation packages offered by several *Service* providers need to be compared by consumers in order to select the best *Service* that meets their requirements. However, for online *Services* the user can either buy a *Service* package or can select simple *Services* individually and compose them. In the former situation, the packages are treated as single simple *Services* and ranked by the X-Algorithm which we have discussed earlier in this thesis. In this Chapter, we discuss the latter situation in which users cannot determine the best *Service* composition because of at least two reasons. One reason is, the *Services* may be obtained from different *Service* providers. The other reason is, to manually rank the composed *Services* based on the preferences of individual *Services* is hard. So, there is a need to provide an automatic algorithm for ranking such compositions. In this Chapter we explain how the X-Algorithm can be used to rank such composite *Services*.

7.1 Request Structure for Simple Services In a Composition

A composition is performed on many *Service* types. In the situation where a user wants to compose hotel accommodation *Services* with airline booking *Services* we regard "hotel booking" and "airline booking" as two *Service* types. So, it is necessary for the user to compose a simple request for ranking simple *Services*, which has been discussed in Section 4.2.2, for each *Service* type. Based upon each request options and semantics, the X-Algorithm will produce a ranked list of available *Services*. In order to automatically combine and produce a ranking for the combinations it is necessary that the user inputs more options along with the simple request that govern the level of preference and importance of each *Service* type. So, we expand the simple request structure, introduced earlier for ranking simple services, with two additional fields Simple Request Weight(SRW) and Simple Request Essential (SRE). This extended structure, shown in Figure 39, is called *Extended Simple Request Structure* ($ESRS$). It is necessary to automatically construct a *Composite Request Structure* (CRS). We explain this construction in the next section. These two fields apply to *all the Services* ranked by the X-Algorithm, as compared to the other fields under *Simple Request* that apply to *attributes of Services* to be ranked by the X-Algorithm. That is, *Simple Request* (SR) field in *ESRS* is the user request to rank simple *Services* by the X-Algorithm that fit the attribute values and semantics defined in it. The value $weight$ defined by the user for the attribute *SRW* in *ESRS* is the weight assigned to

$Simple\ Request(SR)$				SRW	SRE
-	$Feature_1$	...	$Feature_n$	$weight$	ON/OFF
$Query$	$value_1$	...	$value_n$		
$Semantics$	$semantic_1$	...	$semantic_n$		
$Weight$	$weight_1$	...	$weight_n$		
$Mode$	$mode_1$	...	$mode_n$		
$Range$	$range_1$	...	$range_n$		
$Essential$	ess_1	...	ess_n		
$AllBest$	ON/OFF				

Figure 39: The structure of *ESRS* request

all Services ranked in response to the *Simple Request* part. If the attribute *SRE* in *ESRS* is set to ”ON” (”OFF”) then all *Services* ranked by the X-Algorithm is set to “ON” (“OFF”). We emphasize the necessity and importance of these two fields. In these two fields the user is required to assign preference and level of importance (regular or essential) to the *Service* within the composition, which will lead to determining whether a simple *Service* in a composition can be more important than another simple *Service*. Eventually, this will affect the ranks produced for the different compositions.

Thus, a user wishing to select *m* different *Service* sets for a composition will construct a *ESRS* for each *Service* type. Once the *ESRS* for all *m* requests are submitted, the system environment will automatically construct a CRS, rank *Services* based on the SR part of each *ESRS*, apply the CRS on the compositions of the ranked *Service* sets, and produce the rankings of compositions.

7.2 Constructing Composite Request from Extended Simple Requests

Let $RqC = \{Rq_1, Rq_2, \ldots, Rq_n\}$, where Rq_i denote the *ESRS* for i_{th} *Service* type. In the X-Algorithm every attribute will get a bounded score. We need to extend this concept to *Service* level. So, we provide a method for calculating the “maximum rank” *maxRank* for each

Rq_i. The calculation is done based on the regular ranks, and other options of the attributes within Rq_i definition in $ESRS$. So, this maximum rank can be calculated prior to the ranking process. We define $maxRank = MRR + MEss + MAllBest$, where:

- MRR is the maximum rank that the X-Algorithm achieves when both $Essential$ and $AllBest$ options are not set.
- $MEss$ is the maximum rank when $Essential$ option is set to "ON".
- $MAllBest$ is the maximum when $AllBest$ option is set to "ON".

Since MRR, $MEss$, and $MAllBest$ are all bounded, the value $maxRank$ will be bounded.

1. MRR is calculated in the multiplication of the weights by the scores of *Service* features, when all scores are set to the maximum reward ($maxReward$) and all the weights are set to the maximum weight(w_h). Thus, $MRR = n * w_h * maxReward$ (where n is the number of features defined in *Query*).

2. The method to obtain essential weight w_e is explained in Equation 39 in Section 5.5. Thus, $MEss = maxReward * ((w_h * 1000 * w_e) + w_h) * nef$. (where w_h and w_e are the highest regular weight and the essential weight respectively, and nef is the number of essential features defined in the user request).

3. As explained in Section 5.6, the maximum impact caused by this option on the final rank of Rq_i is an addition of the number of features defined in Rq_i to the final rank of the results of Rq_i. Thus, $MAllBest = n * AllBestFlag$, (where n is the number of features in *Query* and $AllBestFlag$ is 0 if $AllBest$ option is "OFF" and $AllBestFlag$ is 1 if $AllBest$ option is "ON").

We include the $maxRank_i$ for $ESRS$ Rq_i in constructing CRS.

Remark. *If our algorithm does not produce bounded output, calculating the maximum value should be after ranking all the Services for each Rq_i, and then analyse the results to find the maximum result for each Rq_i. This highlights another importance of bounded results requirement.*

The X-Algorithm interface is extended so that when it receives the first $ESRS$ structure, it will start constructing the CRS. The CRS contains the following fields: (1) Complex Request Query (CRQ), (2) Complex Request Weight(CRW), (3) Complex Request Essential (CRE). Table 34 shows the CRS for the composite request that involves m $ESRS$ requests.

-	Rq_1	$\dots$	Rq_m
CRQ	$maxRank_{Rq_1}$	$\dots$	$maxRank_{Rq_m}$
CRW	SRW_{Rq_1}	$\dots$	SRW_{Rq_m}
CRE	SRE_{Rq_1}	$\dots$	SRE_{Rq_m}

Table 34: The Structure of CRS request for m $ESRS$.

7.3 Responding to a Composite Request

The X-Algorithm receives m $ESRS$, constructs a CRS, and does the following steps.

1. Rank *Services* for each Simple Request(Rq_i) using the X-Algorithm. Corresponding to Rq_i let TR_i denote the total ranks of *Services* calculated by X-Algorithm.

2. Compute the Cartesian Product for different total ranks, $TR = \{TR_1 \times TR_2 \times \cdots \times TR_m\}$. We represent TR in $CPlan$ matrix as,

$$CPlan = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1j} & \dots & r_{1m} \\ r_{21} & r_{22} & \dots & r_{2j} & \dots & r_{2m} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ r_{i1} & r_{i2} & \dots & r_{ij} & \dots & r_{im} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ r_{C1} & r_{C2} & \dots & r_{Cj} & \dots & r_{Cm} \end{bmatrix} \tag{52}$$

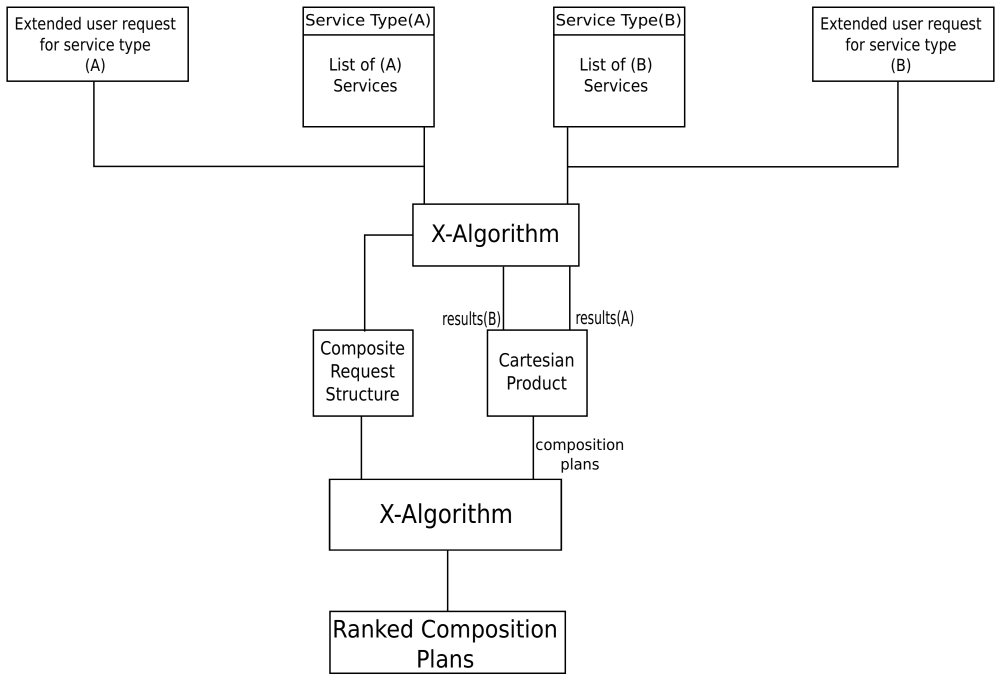


Figure 40: An illustration of the *Service* composition

where $C =| TR |$. Each row in $CPlan$ shows the rankings of simple *Services* in the composition. That is, the i_{th} row is the ranking vector of i_{th} composition, such that $r_{ij} \in TR_j$ is the ranking of the simple *Services* corresponding to Rq_j.

3. We regard the rows of $CPlan$ matrix as alternates and would like to rank them with respect to Exact Match criteria of the ranks specified in CRQ subject to the weights CRW and the $Essential$ option CRE specified Table 34. So, we are justified in running the X-Algorithm considering CRS as the user request and $CPlan$ as the available alternatives.

   Figure 40 illustrates the composing algorithm of the two *Services*, *Service*(A) and *Service*(B).

Example 11. *In this example, we illustrate the steps of ranking Composed Services in practice. Suppose, a user defined one Composed Request that includes two different types of ESRS Rq_h and Rq_f defined in Table 35 and Table 36. For simplicity, we consider both $EssentialAccuracy$ and $AlgorithmAccuracy$ are pre-set to* 0.1*. Also, we assume that Hotel and Air Travel Services are* *defined in Table 37 and Table 38.*

$Simple\ Hotel\ Request(Rq_h)$				SRW_h	SRE_h
-	$HotelPrice(h_1)$	$Stars(h_2)$	$Location(h_3)$	$Significant$	$0(OFF)$
$Query$	100$	4/5	$Downtown$		
$Semantics$	LB	MB	EB		
$Weight$	$Significant$	$Normal$	Low		
$Mode$	$Best$	$Exact$	$-$		
$Range$	$-$	$-$	$-$		
$Essential$	1	0	0		
$AllBest$	$0(OFF)$				

Table 35: *ESRS* for Hotel Service

$Simple\ AirTravel\ Request(Rq_f)$			SRW_f	SRE_f
-	$TicketPrice(f_1)$	$Route(f_2)$	$Normal$	$0(OFF)$
$Query$	1000$	$Direct$		
$Semantics$	LB	EB		
$Weight$	$Significant$	$Normal$		
$Mode$	$Best$	$-$		
$Range$	$-$	$-$		
$Essential$	0	0		
$AllBest$	$1(ON)$			

Table 36: *ESRS* for Air Travel Service

-	$Price(hs_1)$	$Stars(hs_2)$	$Location(hs_3)$
$HotelService_1(HS_1)$	60$	3.5/5	$Suburb$
$HotelService_2(HS_2)$	250$	5/5	$Downtown$
$HotelService_3(HS_3)$	100$	5/5	$Suburb$

Table 37: Available Hotel Services

-	$Ticket\ Price(fs_1)$	$Route(fs_2)$
$AirTravelService_1(FS_1)$	900$	$Indirect$
$AirTravelService_2(FS_2)$	1200$	$Direct$
$AirTravelService_3(FS_3)$	1000$	$Direct$

Table 38: Available Air Travel Services

We perform the following steps:

1. *Calculate the $maxRank_h$ for the Simple Hotel Request Rq_h, and $maxRank_f$ for the Simple*

AirTravel Request Rq_f,

$$maxRank_h = RR_h + Ess_h + AllBest_h$$

$$where,$$

$$MRR_h = nfq * w_h * maxReward = 3 * 0.005 * 3 = 0.045$$

$$MEss_h = maxReward * ((w_e * 1000w_h) + w_h) * neq$$

$$= 3 * ((0.005 * \frac{0.1+(3-1)*(5)}{0.1} + 1) * 5 + 0.005) * 1 = 22.59$$

$$MAllBest = 3 * 0 = 0$$

$$Thus,$$

$$maxRank_h = 22.62$$

Similarly, we calculate the maxRank for the Simple Request, Rq_f,

$$maxRank_f = MRR_f + MEss_f + MAllBest_f = 0.03 + 0 + 2 = 2.03$$

2. *Table 39 shows the CRS request structured automatically by the X-Algorithm for this example.*

-	Rq_h	Rq_f
CRQ	$maxRank_{Rq_h}(22.62)$	$maxRank_{Rq_f}(2.03)$
CRW	$SRW_{Rq_h}(0.005)$	$SRW_{Rq_f}(0.003)$
CRE	$SRE_{Rq_h}(0)$	$SRE_{Rq_f}(0)$

Table 39: The Structure of CRS request for the two Simple Requests Rq_h and Rq_f.

3. *Rank the available Services for each ESRS using the X-Algorithm. The Total Ranks for different Hotel Services* TR_h *and for different Air Travel Services* TR_f *are shown in Table 40 and Table 41.*

$Service$	$Rank$
HS_1	7.42
HS_2	4.19
HS_3	6.28

Table 40: Ranks for available Hotel Services for ESQS Rq_h

$Service$	$Rank$
FS_1	2.009
FS_2	1.804
FS_3	1.724

Table 41: Ranks for available Air Travel Services for ESQS Rq_f

4. *Compute $TR = TR_h \times TR_f$, which is represented as the $CPlan$ matrix as shown in Table 42*

$TR = TR_h \times TR_f$
$[HS_1\ ,\ FS_1]$
$[HS_1\ ,\ FS_2]$
$[HS_1\ ,\ FS_3]$
$[HS_2\ ,\ FS_1]$
$[HS_2\ ,\ FS_2]$
$[HS_2\ ,\ FS_3]$
$[HS_3\ ,\ FS_1]$
$[HS_3\ ,\ FS_2]$
$[HS_3\ ,\ FS_3]$

$$\Longrightarrow CPlan = \begin{bmatrix} HS_{11} & FS_{12} \\ HS_{21} & FS_{22} \\ HS_{31} & FS_{32} \\ HS_{41} & FS_{42} \\ HS_{51} & FS_{52} \\ HS_{61} & FS_{62} \\ HS_{71} & FS_{72} \\ HS_{81} & FS_{82} \\ HS_{91} & FS_{92} \end{bmatrix}$$

Table 42: The different composition plans ($CPlan$) for Hotel and Air Travel Services.

5. *Run the X-Algorithm on the Request CRS and on the available alternatives in $CPlan$. The Ranks of rows in $CPlan$ are listed in Table 43 (ordered in a non-increasing order).*

$Ranked\ TR$	$CPlan\ Ranks$
$[HS_1\ ,\ FS_3]$	0.0048
$[HS_1\ ,\ FS_2]$	0.0042
$[HS_3\ ,\ FS_3]$	0.0041
$[HS_1\ ,\ FS_1]$	0.0040
$[HS_3\ ,\ FS_2]$	0.0035
$[HS_3\ ,\ FS_1]$	0.0032
$[HS_2\ ,\ FS_3]$	0.0029
$[HS_2\ ,\ FS_2]$	0.0023
$[HS_2\ ,\ FS_1]$	0.0021

Table 43: Ranks of the different composition plans based on the Composite Request defined by the user

The results shown in Table 43 are based on the Composite request CRS that does not includes any essential weight. Assume the user defined an essential weight to Hotel Simple Services such that *the X-Algorithm considers first the goodness of the Hotel Services, and then the Air Travel Services in the composition plan. Table 44 shows the request CRS with essential option set to Hotel Service.*

-	Rq_h	Rq_f
CRQ	$maxRank_{Rq_h}(22.62)$	$maxRank_{Rq_f}(2.03)$
CRW	$SRW_{Rq_h}(0.005)$	$SRW_{Rq_f}(0.003)$
CRE	$SRE_{Rq_h}(1)$	$SRE_{Rq_f}(0)$

Table 44: The Structure of CRS request for the two $ESRS$ requests Rq_h and Rq_f.

The results of this request is listed in the following table(ordered in a non-increasing order). The ordering is essentially based on the Hotel Simple Services. The composite Services with a specific Hotel name ordered on Air Travel Services.

$ComposedServices$	$Composition\ Ranks$
$[HRS_1\ ,\ FRS_3]$	2.3381
$[HRS_1\ ,\ FRS_2]$	2.3375
$[HRS_1\ ,\ FRS_1]$	2.3373
$[HRS_3\ ,\ FRS_3]$	1.3872
$[HRS_3\ ,\ FRS_2]$	1.3866
$[HRS_3\ ,\ FRS_1]$	1.3863
$[HRS_2\ ,\ FRS_3]$	-0.0632
$[HRS_2\ ,\ FRS_2]$	-0.0638
$[HRS_2\ ,\ FRS_1]$	-0.0641

7.4 The Complexity of Ranking Composed Services

Let Rq_i denote the i_{th} $ESRS$ in a Composite Request CRS. Let there be m number of Simple Requests in CRS. The size of composition plan is C.

Based on the Complexity of X-Algorithm shown in Section 6.2, the time necessary to calculate the ranks of $CPlan$ is shown below

$$Time_{Composed\ X\_Algo} = O(C) + O(C * m) + O(C\ log(C))$$

where, $O(C)$ is the time necessary to obtain composition plans, $O(C*m)$ is the time necessary to calculate the total ranks for m composition plans in $CPlan$, and $O(C\ log(C))$ is the time necessary to sort the total ranks of $CPlan$.

We remark that C is essentially dependant on the number of *Services* ranked and composed by the X-Algorithm and m is the number of *ESRS* requests. There is no obvious correlation between C and m. For a fixed value of m, the complexity is bounded by $O(C\ log(C))$.

7.5 Summary

In this chapter we introduced the Composed Query Option for the X-Algorithm. We have illustrated the steps taken by the X-Algorithm to rank and respond to Composed Queries. Finally, we introduced the Algorithmic Complexity associated with this option.

Chapter 8

Conclusion and Future Work

In this thesis we have developed a solution to rank *Services* that have many heterogeneous features. Our algorithm contribute to the field of Service-Oriented Computing. Also, it provides a generic solution to any ranking problem in which input and alternatives for ranking can be described as vectors.

The X-Algorithm satisfies the fairness criteria, the scale set for comparing ranking algorithms. The input to the algorithm is a request that includes query and preferences from the user. These are applied against a set of *Services* in the system. The *Services* that match a query and comply with the semantics preferences that are specified in the user request are selected and ranked. X-Algorithm computes the scores for each *Service* attribute and a total rank for the whole *Service* with respect to the user request. The selected *Services* are ranked based on the total score. X-Algorithm has been extended to rank composite *Services* based on simple extensions to user requests. Thus, X-Algorithm is both user-centric and semantic-based. The semantics allow users to incorporate a wide range of options and preferences that are normally done informally in daily life for ranking *Services*. These options include *Ranking modes, Essential, AllBest, Range* and *Service composition*. Generally, options may be combined. There exist exceptions that are indicated in Table 45

Options	ExactMode	BestMode	Essential	AllBest	Range
ExactMode	■	Yes	Yes	Yes	Yes
BestMode	Yes	■	Yes	Yes	Yes
Essential	Yes	Yes	■	No*	Yes
AllBest	Yes	Yes	No*	■	Yes
Range	No*	Yes	No*	Yes	■

Table 45: This Table shows the ability/inability to integrate the X-algorithm options together

*– Essential option cannot be combined with AllBest option because both of them perform some adjustments on the ranking concept
– *Exact Mode* looks for exact match to one value, and Range option by default looks for numbers exact to ones within the range, therefore, they cannot be combined

8.1 Evaluation

We have evaluated the X-Algorithm computational level, where we conducted different experiments to evaluate the performance and accuracy of the X-Algorithm in practical settings. To validate the accuracy, we tested the behaviour of the algorithm on a case study. We tried different combinations of options and under each set of options the algorithm behaved exactly as expected. A manual inspection revealed the satisfaction of user specified semantics in all outcomes. We tested the run time efficiency of the algorithm on randomly generated data sets. In these practical studies, we found only a linear expansion of run time. In this section we evaluate our algorithm at a conceptual level. That is, we evaluate the X-Algorithm in terms of its satisfaction to the consumer and algorithmic requirements we defined for fair ranking. Table 46 and Table 47 show this evaluation.

As shown in Table 46 and Table 47, the X-Algorithm satisfies all the fair ranking requirements. However, we did not compare our algorithm with other algorithms. This is because there exists no algorithm with which it is meaningfully useful to compare.

Fair Ranking Requirements	X-Algorithm	Explanation
Ability to find better values	$\surd$	By using Best Mode option
Numerical/non-numerical values	$\surd$	X-Algorithm accepts numbers, strings and boolean datatypes.
Online ranking without assumptions	$\surd$	Outputs are generated on-spot based on user's input only.
Options to manipulate results	$\surd$	X-Algorithm offers many options to users: Best/Exact Modes, Range, AllBest, Essential and Semantic choice.
Fast response	$\surd$	The practical evaluation shows linear growth of the algorithms runtime.
User-friendly *Query* building	$\surd$	Options can be easily integrated and structured in a simple user-interface.

Table 46: Evaluation of the X-Algorithm with respect to consumer perspective requirements

Fair Ranking Requirements	X-Algorithm	Explanation
Different semantics support	$\surd$	X-Algorithm considers MB,LB and EB semantics
Normalized outputs	$\surd$	The X-Algorithm is based on the *Similarity Measure* SM_X which produces normalized outputs
Consistent outputs	$\surd$	Outputs are based on inputs at any given time. Identical inputs produce Identical outputs.
Input range $(0, \infty)$	$\surd$	X-Algorithm puts no restrictions on inputs.
Unlimited number of features	$\surd$	Number of features is unlimited.
Built on a simple concept	$\surd$	X-Algorithm is based on Vector-Based *Similarity Measures* which are one of the simplest and easiest ranking approaches.

Table 47: Evaluation of the X-Algorithm with respect to consumer perspective requirements

8.2 Future Work

In order to realize the full potential of the X-Algorithm it should be properly plugged in a context-based SOA architecture as shown in Figure 41. This figure shows the interaction scenario among the SOA components that jointly enable *Service* publication, *Service* discovery, *Service* ranking and execution. The results of this thesis, when combined with Alaa Alsaig

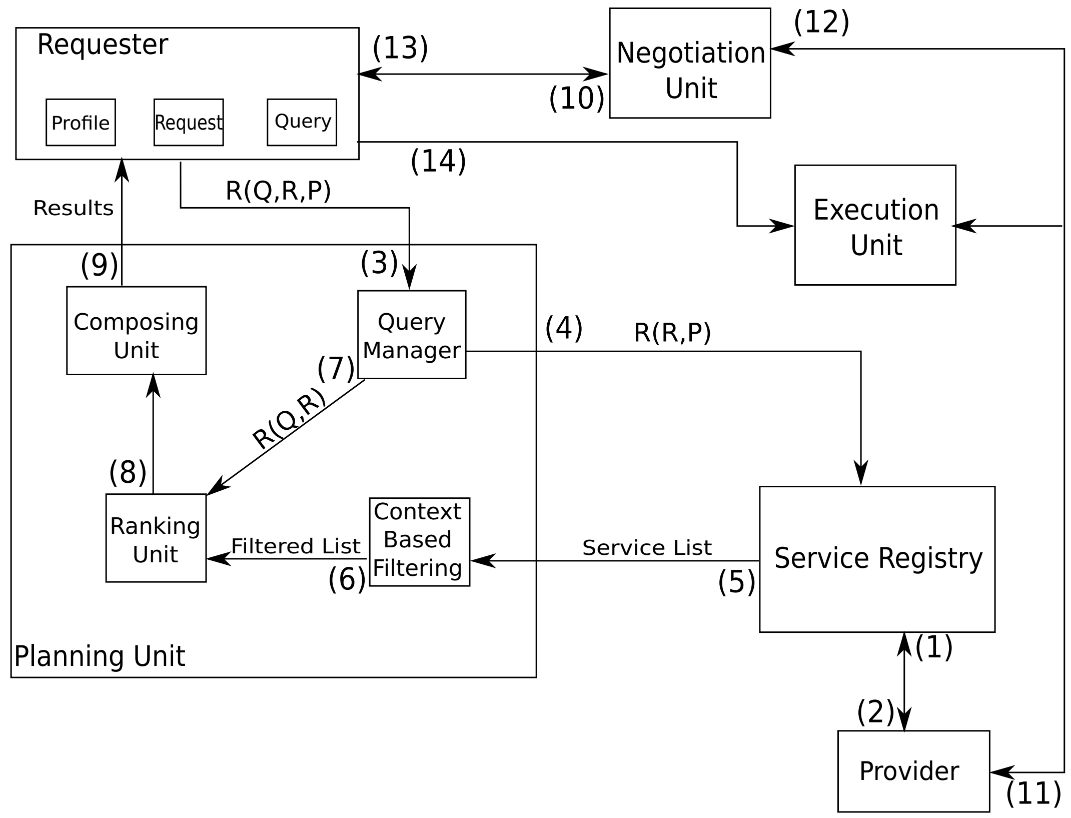


Figure 41: The Ranking Algorithm in FrSeC Framework

[Als13], who has just completed the design and implementation of the *Service* registry using NoSQL technology, will cover the major tasks in Figure 41. The natural next step is to implement the following units of Figure 41.

- *Planning Unit* with interfaces to the *Service* registry, and the query processing unit *Query Manager*,
- a user interface for service requesters to compose queries with their semantic preferences, and
- *Composing unit*, *Negotiating Unit*, *Execution Unit*.

This is an important essential part of future work.

In order to be convinced of the operational ability of SOA in in Figure 41, we have numbered the arrows in the figure in the order in which the tasks are to be performed, from publication to execution. *Service* publication process is done in (1,2), and is achieved by the thesis work of Alaa Alsaig [Als13]. A service requester interacts (3) with *Query Manager* who shares relevant information with *Service Registry*(4) and *Ranking Unit* (7). The selected *Services* from the *Service Registry* are sent to *Context-based Filtering* unit (5), which upon filtering are sent to *Ranking Unit* (6). Both *Query Manager* and *Context-based Query Manager* units are trivial designs, because they perform a simple task. The X-Algorithm is applied in *Ranking Unit* and the results are sent to the *Composing Unit* (8). *Service* composition is necessary only for composite queries. The results are provided to the user (9). In selection process the user has two options, either to have the *Service* executed (14), or send a negotiation request to the *Negotiation Unit* (10). In the later case, the negotiation request is forwarded to the service provider (11) who responds to the request (12). The *Negotiation Unit* sends the response to the user (13). This process can go on until either both parties agree or the service requester terminates the session. In the former case the service requester requests an execution of the *Service* (14). The responsibility of the *Execution Unit* to assure the complete delivery of the *Service* to the requester.

Another important aspect of future work is related to applying the X-Algorithm for problems that require vector-based ranking. The potential avenues to look at are ranking for admission to educational institutions (such as medical schools) and for preference-based ranking of companions in social networks.

Bibliography

[AE07] F. Gregory Ashby and D. M. Ennis. Similarity measures. *Scholarpedia*, 2(12):4116, 2007.

[Aga11] S. Agarwal. The infinite push: A new support vector ranking algorithm that directly optimizes accuracy at the absolute top of the list. In *Proceedings of the SIAM International Conference on Data Mining*, 2011.

[Als13] Alaa Alsaig. Context-aware service registry modeling and implementation. Master's thesis, Concordia University, 2013.

[AT05] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[BB08] J.O. Bennett and W.L. Briggs. *Using and understanding mathematics: A quantitative reasoning approach*. Pearson Addison Wesley, 2008.

[BM08] J.A. Bondy and U.S.R. Murty. *Graph theory*. Berlin: Springer, 2008.

[Cha07] S.H. Cha. Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1, 2007.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[CYT05] S.H. Cha, S. Yoon, and C.C. Tappert. Enhancing binary feature vector similarity measures. 2005.

[Dan80] P.E. Danielsson. Euclidean distance mapping. *Computer Graphics and image processing*, 14(3):227–248, 1980.

[DX08] H. Dinh and L. Xu. Measuring the similarity of vector fields using global distributions. *Structural, Syntactic, and Statistical Pattern Recognition*, pages 187–196, 2008.

[Erl04] Thomas Erl. *Service-oriented architecture*. Prentice Hall Englewood Cliffs, 2004.

[GHTH11] T. Gwo-Hshiung, G.H. Tzeng, and J.J. Huang. *Multiple Attribute Decision Making: Methods and Applications*. CRC Press, 2011.

[GJG04] L.A. Granka, T. Joachims, and G. Gay. Eye-tracking analysis of user behavior in www search. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 478–479. ACM, 2004.

[GVB12] S.K. Garg, S. Versteeg, and R. Buyya. A framework for ranking of cloud computing services. *Future Generation Computer Systems*, 2012.

[Han09] C.W. Hang. Trustworthy service-oriented computing. *Doctoral Mentoring Program*, page 7, 2009.

[HL97] Duane Hanselman and Bruce C Littlefield. *Mastering MATLAB 5: A comprehensive tutorial and reference*. Prentice Hall PTR, 1997.

[Ibr12] Naseem Ismail Ibrahim. *Specification, Composition and Provision of Trustworthy Context-dependent Services*. PhD thesis, Computer Science and Software Eng., Concordia University, 2012.

[Inc08] Google Inc. Google Play Application Store. `https://play.google.com/`, 2008.

[Joa02] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

[JRVF09] G. Jurman, S. Riccadonna, R. Visintainer, and C. Furlanello. Canberra distance on ranked lists. In *Proceedings, Advances in Ranking–NIPS 09 Workshop*, pages 22–27, 2009.

[Kah08] C. Kahraman. Multi-criteria decision making methods and fuzzy sets. *Fuzzy Multi-Criteria Decision Making*, pages 1–18, 2008.

[LHB$^+$08] L. Lorigo, M. Haridasan, H. Brynjarsdóttir, L. Xia, T. Joachims, G. Gay, L. Granka, F. Pellacini, and B. Pan. Eye tracking and online search: Lessons learned and challenges ahead. *Journal of the American Society for Information Science and Technology*, 59(7):1041–1052, 2008.

[LM98] Huan Liu and Hiroshi Motoda. *Feature selection for knowledge discovery and data mining*. Springer, 1998.

[LY93] H-J Li and S-H Yang. Using range profiles as feature vectors to identify aerospace objects. *Antennas and Propagation, IEEE Transactions on*, 41(3):261–268, 1993.

[Mih04] R. Mihalcea. Graph-based ranking algorithms for sentence extraction, applied to text summarization. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 20. Association for Computational Linguistics, 2004.

[MMM12] Ana Milovanović, Maja Mitričević, and Anđela Mijalković. The analytic hierarchy process (ahp) application in equipment selection. *THE GROWTH OF SOFTWARE*

INDUSTRY IN THE WORLD WITH SPECIAL FOCUS ON BOSNIA AND HERZEGOVINA.. 1880, page 1912, 2012.

[OH11] K. Oku and F. Hattori. Fusion-based recommender system for improving serendipity. In *Proceedings of the Workshop on Novelty and Diversity in Recommender Systems (DiveRS 2011), at the 5th ACM International Conference on Recommender Systems (RecSys 2011)*, page 19, 2011.

[PHJ+07] B. Pan, H. Hembrooke, T. Joachims, L. Lorigo, G. Gay, and L. Granka. In google we trust: Users' decisions on rank, position, and relevance. *Journal of Computer-Mediated Communication*, 12(3):801–823, 2007.

[PS09] Selwyn Piramuthu and Riyaz T Sikora. Iterative feature construction for improving inductive learning algorithms. *Expert Systems with Applications*, 36(2):3401–3406, 2009.

[RB09] K. Riesen and H. Bunke. Feature ranking algorithms for improving classification of vector space embedded graphs. In *Computer Analysis of Images and Patterns*, pages 377–384. Springer, 2009.

[Saa83] Thomas L Saaty. Priority setting in complex problems. *Engineering Management, IEEE Transactions on*, (3):140–155, 1983.

[Saa08] Thomas L Saaty. Decision making with the analytic hierarchy process. *International Journal of Services Sciences*, 1(1):83–98, 2008.

[SB09] Julia Sidorova and Toni Badia. Syntactic learning for eseda. 1, a tool for enhanced speech emotion detection and analysis. In *Internet Technology and Secured Transactions, 2009. ICITST 2009. International Conference for*, pages 1–6. IEEE, 2009.

[SH06] Munindar P Singh and Michael N Huhns. *Service-oriented computing: semantics, processes, agents*. Wiley. com, 2006.

[Sig05] A. Signorini. A survey of ranking algorithms. 2005.

[SKR99] J.B. Schafer, J. Konstan, and J. Riedi. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166. ACM, 1999.

[T+77] Amos Tversky et al. Features of similarity. *Psychological review*, 84(4):327–352, 1977.

[Tek06] K. Teknomo. Analytic hierarchy process (ahp) tutorial. *Kardi Teknomo's page*, 2006.

[WMZ10] W. Webber, A. Moffat, and J. Zobel. A similarity measure for indefinite rankings. *ACM Transactions on Information Systems (TOIS)*, 28(4):20, 2010.

[XNJR02] Eric P Xing, Andrew Y Ng, Michael I Jordan, and Stuart Russell. Distance metric learning, with application to clustering with side-information. *Advances in neural information processing systems*, 15:505–512, 2002.

[ZWQZ11] Y. Zhu, J. Wen, M. Qin, and G. Zhou. Web service selection mechanism with qos and trust management. *Journal of Information Computational Science*, 8(12):2327–2334, 2011.

Appendix A

Symmetry Calculations For Similarity Measures

This section is to prove the absence of symmetry characteristics in the SMs introduced in the literature survey.

Consider the following assumptions:

- each SM is tested with two different inputs S_1 and S_2, which are symmetric around q.
- the similarity measure is considered symmetric, if it produces similar outputs for S_1 and S_2, and asymmetric otherwise.
- value of S_2 is an image of value of S_1 around q, which means that $q - S_1 = S_2 - q$ where $S_1 < q$ and $S_2 > q$
- assume the value of $S_1 = a$, where $a < q$ and since S_2 is an image of S_1 around q, $S_2 = q - a + q = 2q + a$ which means that $2q + a > q$, FIGURE(***).
- we want to prove that $SM(S_1) = SM(S_2)$ for each SM in Tables 1,2.
- LS denotes left side of equation and RS denotes right side, which means that we want to

prove that LS=RS where LS is the SM with S_1 as input and RS is the same SM with S_2 as input.

- we assume S_1 and S_2 have only one value, so, any sum is removed. In other words, since we are not using vector of values, there is no need to calculate the total ranks for different elements of vectors, thus, sum is removed from SMs.

1. Euclidean Distance,

$$LS = d_{Euc}(S_1) = \sqrt[2]{|S_1 - q|^2} = \sqrt[2]{|a - q|^2}$$

$$RS = d_{Euc}(S_2) = \sqrt[2]{|S_2 - q|^2} = \sqrt[2]{|(2q - a) - q|^2} = \sqrt[2]{|q - a|^2} = LS$$

Hence, Euclidean Distance is a symmetric SM.

2. Kulczynski

$$LS = d_{kul}(S_1) = \frac{|S_1 - q|}{min(S_1, q)} = \frac{|a - q|}{a}$$

$$RS = d_{kul}(S_2) = \frac{|S_2 - q|}{min(S_2, q)} = \frac{|(2q - a) - q|}{q} = \frac{|q - a|}{q} \neq LS$$

Thus, Kulczynski is an asymmetric SM

3. Gower

$$LS = d_{gow}(S_1) = |S_1 - q| = |a - q|$$

$$RS = d_{gow}(S_2) = |S_2 - q| = |2q - a - q| = |q - a| = LS$$

As a result Gower is a symmetric SM.

4. Lorentzian

$$LS = d_{lor}(S_1) = \ln(1 + |S_1 - q|) = \ln(1 + |a - q|)$$

$$RS = d_{lor}(S_2) = \ln(1 + |S_2 - q|) = \ln(1 + |2q - a - q|) = \ln(1 + |q - a|) = LS$$

So, Lorentzian is a symmetric SM.

5. Inner Product

$$LS = d_{IP}(S_1) = S_1 q = aq$$

$$RS = d_{IP}(S_2) = S_2 q = (2q - a) * q = 2q^2 - aq \neq LS$$

Hence, Inner Product is an asymmetric SM.

6. Sørensen

$$LS = d_{sor}(S_1) = \frac{|S_1 - q|}{(S_1 + q)} = \frac{|a - q|}{(a + q)}$$
$$RS = d_{sor}(S_2) = \frac{|S_2 - q|}{(S_2 + q)} = \frac{|q - a|}{(3q - a)} \neq LS$$

Thus, Sørensen is an asymmetric SM.

7. Soergel

$$LS = d_{sg}(S_1) = \frac{|S_1 - q|}{max(S_1, q)} = \frac{|a - q|}{q}$$
$$RS = d_{sg}(S_2) = \frac{|S_2 - q|}{max(S_2, q)} = \frac{|q - a|}{2q - a} \neq LS$$

As a result, Soergel is an asymmetric SM.

8. Cosine

$$LS = S_{cos}(S_1) = \frac{S_1 q}{\sqrt{S_1^2}\sqrt{q^2}} = \frac{aq}{\sqrt{a^2}\sqrt{q^2}}$$
$$RS = S_{cos}(S_2) = \frac{S_2 q}{\sqrt{S_2^2}\sqrt{q^2}} = \frac{2q^2 - aq}{\sqrt{(2q - a)^2}\sqrt{q^2}} \neq LS$$

So, Cosine is an asymmetric SM.

9. Canberra

$$LS = d_{can}(S_1) = \frac{|S_1 - q|}{S_1 + q} = \frac{|a - q|}{a + q}$$
$$RS = d_{can}(S_2) = \frac{|S_2 - q|}{S_2 + q} = \frac{|q - a|}{3q - a} \neq LS$$

Hence, Canberra is an asymmetric SM.

10. Jaccard

$$LS = d_{jac}(S_1) = \frac{(S_1 - q)^2}{S_1^2 + q^2 - S_1 q} = \frac{(a-q)^2}{a^2 + q^2 - aq}$$

$$RS = d_{jac}(S_2) = \frac{(S_2 - q)^2}{S_2^2 + q^2 - S_2 q} = \frac{(q-a)^2}{(2q-a)^2 + q^2 - 2q^2 - aq} \neq LS$$

Thus, Jaccard is an asymmetric SM.

11. Harmonic Mean

$$LS = d_{HM}(S_1) = 2\frac{S_1 q}{S_1 + q} = 2\frac{aq}{a+q}$$

$$RS = d_{HM}(S_2) = 2\frac{S_2 q}{S_2 + q} = 2\frac{2q^2 - aq}{3q - a} \neq LS$$

As a result, Harmonic Mean is an asymmetric SM.

12. Dice

$$LS = d_{dice}(S_1) = \frac{(S_1 - q)^2}{S_1^2 q^2} = \frac{(a-q)^2}{a^2 q^2}$$

$$RS = d_{dice}(S_2) = \frac{(S_2 - q)^2}{S_2^2 q^2} = \frac{(q-a)^2}{(2q-a)^2 q^2} \neq LS$$

Hence, Dice is an asymmetric SM.

13. Relative Change

$$LS = d_{RC}(S_1) = \frac{|S_1 - q|}{max(S_1, q)} = \frac{|a-q|}{q}$$

$$RS = d_{RC}(S_2) = \frac{|S_2 - q|}{max(S_2, q)} = \frac{|q-a|}{(2q-a)} \neq LS$$

Thus, Relative Change is an asymmetric SM.

14. Service Ranking

$$LS = d_{RS}(S_1) = 2 - \frac{S_1}{q}$$

$$RS = d_{RS}(S_2) = 2 - \frac{S_2}{q}$$

since $S_1 \neq S_2$, then, $LS \neq RS$. Thus, Ranking Service is an asymmetric SM.

Appendix B

Limits Calculation For Similarity Measures

In order to be able to calculate the limit of each similarity measure we have to make the following assumptions

- Since the value of query is the same for all services, we can consider the query as a constant denoted by c.
- Number of properties has to be limited. Hence, we consider it as a constant too denoted by c_1.
- Since the changing part is the value of the service's property, we will denote it as x and substitute it by $+\infty$
- Since the expected inputs are lower-bounded by zero and does not include minus values, we will exclude the calculation of $\lim_{x \to -\infty}$
- Since the upper-bound is all what matters for us, we won't bother looking at $\lim_{x \to 0}$.

1. Euclidean Distance

$$\lim_{x\to\infty} \sqrt[2]{|x-c|^2}$$

because of the absolute function, we know it is a positive value, thus,

$$\lim_{x\to\infty} |x-c| = |\lim_{x\to\infty} x - \lim_{x\to\infty} c| = \infty$$

2. Kulczynski

$$\lim_{x\to\infty} \frac{|x-c|}{min(x,c)} = \lim_{x\to\infty} \frac{|x-c|}{c} = \frac{|\lim_{x\to\infty} x - \lim_{x\to\infty} c|}{\lim_{x\to\infty} c} = \frac{\infty - c}{c} = \infty$$

3. Gower

$$\lim_{x\to\infty} \frac{1}{c_1}|x-c| = \frac{|\lim_{x\to\infty} x - \lim_{x\to\infty} c|}{\lim_{x\to\infty} c_1} = \frac{\infty - c}{c_1} = \infty$$

4. Lorentzian

$$\lim_{x\to\infty} \ln(1+|x-c|) = \lim_{x\to\infty} \ln(1+|\infty-c|) = \lim_{x\to\infty} \ln(\infty) = \infty$$

5. Sørensen

$$\lim_{x\to\infty} \frac{|x-c|}{(x+c)} = \frac{\infty - c}{\infty + c} = \frac{\infty}{\infty}$$

if we plug in ∞ we end up with an indeterminate form $\frac{\infty}{\infty}$, thus,

$$\lim_{x\to\infty} \frac{|x-c|}{(x+c)} \times \frac{1/x}{1/x} = \lim_{x\to\infty} \frac{|1-c/x|}{1+c/x} = \frac{1-0}{1+0} = 1$$

6. Soergel

$$\lim_{x\to\infty} \frac{|x-c|}{max(x,c)}$$

since x is ∞ it is larger than c, so we can say,

$$\lim_{x\to\infty} \frac{|x-c|}{x}$$

this leads us to an indeterminate form $\frac{\infty}{\infty}$, using the same method used above,

$$\lim_{x\to\infty} \frac{|x-c|}{x} \times \frac{1/x}{1/x} = \frac{1-0}{1} = 1$$

7. Cosine

$$\frac{xc}{\sqrt{x^2}\sqrt{c^2}}$$

this clearly leads to indeterminate form $\frac{\infty}{\infty}$, thus

$$\lim_{x\to\infty} \frac{xc}{\sqrt{x^2}\sqrt{c^2}} \times \frac{1/x}{1/x} = \frac{c}{\sqrt{c^2}} \text{ assuming c is a positive number, hence}$$

$$\lim_{x\to\infty} \frac{c}{c} = 1$$

8. Canberra (same case has been addressed in SM 5)

$$\lim_{x\to\infty} \frac{|c-x|}{x+c} = 1$$

9. Jaccard

$$\lim_{x\to\infty} \frac{(x-c)^2}{x^2+c^2-xc}$$

if we try plug-in method, we end up with the combined indeterminate form $\frac{\infty}{\infty-\infty}$, thus, by expanding the nominator we get,

$$\lim_{x\to\infty} \frac{x^2-2xc+c^2}{x^2+c^2-xc} \times \frac{1/x^2}{1/x^2} = \lim_{x\to\infty} \frac{1-(2c/x)+(c^2/x^2)}{1+(c^2/x^2)-(c/x)} = \frac{1-0+0}{1+0-0} = 1$$

10. Harmonic Mean

$$\lim_{x\to\infty} 2\frac{xc}{x+c} \text{ leads to the form } \frac{\infty}{\infty}$$

hence,

$$\lim_{x\to\infty} 2\frac{xc}{x+c} \times \frac{1/x}{1/x} = \lim_{x\to\infty} \frac{2c}{1+c/x} = 2c$$

this means that the limit is based on the value of the query, this is clearly shown in Figure 12 where the query value is 120.

11. 1-Dice

$$1 - \lim_{x\to\infty} \frac{(x-c)^2}{x^2c^2} \quad \text{a plug-in clearly leads to the form } \frac{\infty}{\infty}$$

thus,

$$1 - \lim_{x\to\infty} \frac{x^2-2xc+c^2}{x^2c^2} \times \frac{1/x^2}{1/x^2} = 1 - \lim_{x\to\infty} \frac{1-(2c/x)+(c^2/x^2)}{c^2} = 1 - \frac{1}{c^2}$$

12. Angular Disparity

$$\lim_{x\to\infty} 1 - \frac{\cos^{-1}(cosine)}{\pi}$$

since the limit of cosine when x approaches to infinity is 1 as shown in SM 7, then,

$$\lim_{x\to\infty} 1 - \frac{\cos^{-1}(1)}{\pi} = 1 - \frac{0}{\pi} = 1$$

13. Relative Change is similar to Soergel, which is bounded by 1

14. Service Ranking

$$2 - \lim_{x\to\infty} \frac{x}{c}$$

using plugin method we find that

$$2 - \infty = -\infty$$