POLITECNICO DI MILANO

Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Master of Science in Computer Engineering

Diversification for Multi-Domain Result Sets

Supervisor: Prof. Piero Fraternali

Master Graduation Thesis by: Muhammad Shoaib Iqbal
Student Id. Number 751699

Academic Year 2009/2011

POLITECNICO DI MILANO

Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Corso di Laurea Specialistica in Ingegneria Informatica

Diversification for Multi-Domain Result Sets

Relatore: Prof. Piero Fraternali

Tesi di laurea di: Muhammad Shoaib Iqbal
Matricula 751699

Anno Accademico 2009/2011

Sommario

Dove posso trovare un albergo a Milano, vicino ad una sala concerti, un museo e un buon ristorante?

Queste informazioni sono disponibili sul Web, ma nessun sistema software è in grado di accettare tali query, né di calcolare la risposta. Il search computing si puo definire come una nuova scienza multidisciplinare che fornisce le astrazioni, le fondamenta, i metodi e gli strumenti necessari per rispondere a queste e molte domande simili.

A causa della natura combinatoria dei risultati, le istanze di entità rilevanti (per esempio, alberghi a cinque stelle) tendono a comparire più volte tra le prime combinazioni prodotte in risposta a una query multi-dominio. Per migliorare la qualità dei risultati, è importante equilibrare pertinenza (cioè, valori elevati della funzione di rango) con diversità, che promuove entità diverse, ma quasi egualmente pertinenti nei tra le prime combinazioni. Questa tesi esplora tre differenti nozioni di diversità (categoriale, quantitativa e la mescolanza di categorica e quantitativa) per risultati di query multidominio, confronta sperimentalmente algoritmi alternativi per il trade-off tra la pertinenza e la diversità, ed esegue uno studio con utenti per valutare l'utilità della diversificazione di query multi-dominio.

Abstract

Find a hotel in Milan close to a concert venue, a museum and a good restaurant?

This information is available on the Web, but no software system can accept such queries nor compute the answer. Search computing as the new multi-disciplinary science which provides the abstractions, foundations, methods, and tools required to answer these and many similar questions.

Due to the combinatorial nature of results, good entity instances (e.g., five stars hotels) tend to appear repeatedly in top-ranked combinations. To improve the quality of the result set, it is important to balance relevance (i.e., high values of the ranking function) with diversity, which promotes different, yet almost equally relevant, entities in the top-k combinations. This thesis explores three different notions of diversity (categorical, quantitative and mixture of categorical and quantitative) for multi-domain result sets, compares experimentally alternative algorithms for the trade-off between relevance and diversity, and performs a user study for evaluating the utility of diversification in multi-domain queries.

Acknowledgement

This work is fully supported by the Search Computing (SeCo).

I owe most of my gratitude to my Professor Piero Fraternali, who gave me the opportunity to work with him for this thesis during my “Master of Science in Computer Engineering”. I very much appreciate his help and guidance during my whole duration of thesis.

I also want to say thanks to Alessandro Bozzon and Chiara Pasini from the Department of Electronics and Informatics, Politechnico di Milano for providing such a stimulating and friendly environment.

Finally, I am most grateful to my parents for supporting me throughout all these years and for letting me choose my own destination.

Muhammad Shoaib Iqbal

Table of Contents

List of Figure

List of Table

Chapter 1. Introduction

Search Computing is a paradigm for composing search services. While state-of-art search systems answer generic or domain-specific queries, Search Computing enables answering questions via a constellation of dynamically selected, cooperating search services, which are correlated by means of join operations. The idea is simple, yet pervasive. New language and description paradigms are required for expressing queries and for connecting services. New user interfaces and protocols help capturing ranking preferences and enabling their refinement. [1]

1.1 Beyond Page Search

Throughout the last decade, Internet search has been primarily performed by routing users towards the specific Web page that best answered their information needs. Major search engines, such as Google, Yahoo and Bing crawl the Web and index Web pages, highlighting worldwide candidate "best" pages with excellent precision and recall; such ability has proven adequate to fulfill user's needs, to the point that Web search is customarily performed by millions of users, both for work and leisure.

However, not all information needs can be satisfied by individual pages on the surface Web. On one hand, the so-called "deep Web" contains information which is perhaps more valuable than what can be crawled on the surface Web; on another side, as the users get confident in the use of search engines, their queries become more and more complex, to the point that their formulation goes beyond what can be expressed with a few keywords, their answers require more than a list of Web pages, and general-purpose search engines perform poorly upon them. According to search company's experts, the number of complex queries that are not answered well by major search engines due to their intrinsic complexity is remarkably high and increasing. Many search interactions can be considered as part of a more complex process of expressing goals and achieving tasks.

When a query addresses a specific domain (e.g., travels, music, shows, food, movies, health, and genetic diseases), domain-specific search engines do a better job than general-purpose ones; but their expertise is focused upon a given domain. Thus, one can separately find best travel offers and interesting music shows, or conduct genetic analysis and investigate the related medical

literature, but can hardly combine information from diverse yet related domains. An expert user can perform several independent searches and then manually combine the findings, but such procedure is cumbersome and error prone.

Search Computing aims at responding to multi-domain queries, i.e., queries over multiple semantic fields of interest, by helping users (or by substituting to them) in their ability to decompose queries and manually assemble complete results from partial answers; thus, Search Computing aims at filling the gap between generalized search systems, which are unable to find information spanning multiple topics, and domain-specific search systems, which cannot go beyond their domain limits.

Paradigmatic examples of Search Computing queries are: “Where can I attend an interesting scientific conference in my field and at the same time relax on a beautiful beach nearby?”, “Where is the theatre closest to my hotel, offering a high rank action movie and a near-by pizzeria?”, “Who are the strongest candidates in Europe for competing on software ideas?”, “Who is the best doctor who can cure insomnia in a nearby public hospital?”, “Which are the highest risk factors associated with the most prevalent diseases among the young population?” These examples show that Search Computing aims at covering a large and increasing spectrum of user’s queries, which structurally go beyond the capabilities of general-purpose search engines. These queries cannot be answered without capturing some of their semantics, which at minimum consists in understanding their underlying domains, in routing appropriate query subsets to each domain specific source and in combining answers from each expert to build a complete answer that is meaningful for the user. [1]

1.2 State of the Art

Processing queries on multiple search engines is not new; meta-search engines are capable of routing the same query to multiple search engines and then presenting composite results. Kosmix is a new-generation meta-search engine connecting to over a thousand of sources by using their Web services. In Kosmix, the relevant data sources for a query are determined by matching the user’s keywords with a huge private concept taxonomy (of about a million nodes), after manually tagging the data sources with the same taxonomy concepts. Kosmix, then, routes the query to all data sources, without attempting source integration. Results are collected from Google, Yahoo,

Flicker, YouTube, Twitter, and so on, and presented to users; sources typically include very popular search engines, but sometime also domain-specific sources, such as the Day-Life or Slate (in the news domain). While Kosmix has the ability of showing individual results from many distinct data sources, it doesn't integrate multiple domains, and therefore cannot answer complex queries; rather, it can answer simple queries by retrieving data from a plurality of sources. Yet, Kosmix demonstrates that Web services are viable methods for getting information from remote sources.

Vertical search engines are focused upon a single domain, e.g., hotels (Booking) or flights (Tuifly), which are well-understood in terms of data quality and ranking criteria (e.g., for hotels and flights ranking depends on price, plus other domain specific criteria, such as the hotel's location and stars, or the flight's duration and number of intermediate stops). Therefore, vertical search engine systems perform o the ranking of search results by using a single scoring function and compute "global best" results for their domain. Compared to Kosmix, these systems are focused upon one single domain of expertise, but they compute "global" rankings and then order their results according to such ranking; instead, Kosmix collates results according to data source relevance, without intermixing items from the various sources. Also vertical search engines use Web services for connecting to data sources, although the number of data services available to a given engine is normally small.

Going beyond meta-search and vertical engines, we find extensions of vertical search engines capable of integrating information from multiple, but "contiguous", domains. For instance, Expedia or Lastminute are capable of integrating information about flights, hotels, car rentals, special events offerings, and so on. The fact that a user selects a flight with given associated departure/arrival times helps the system in proposing the proper length of the hotel stay or of the car rental, thus checking availability and prices and presenting a "best offer". Users normally are well aware of their travelling needs, therefore they select the trip first, and then acquire additional services; however, if they are offered a particularly attractive hotel booking, they can fix that choice and go back to flights, trying to improve their travel plan. Thus, an expert user can combine several travel services and work with combinations, improving each offer separately while maintaining them "connected" to the travel plan, and thus achieving an optimal "global offer". The notion of combination, based upon given destinations, dates and times, is very similar to the notion of composition that we want to develop in Search Computing; in a

sense, Expedia and Lastminute are examples of Search Computing systems, however with given fixed domains and composition patterns.

Advancing search by using knowledge is raising a lot of interest. Knowledge based search systems, such as Yago, Wolfram-Alpha and True Knowledge, work by first building large ontologies and then translating user's queries into requests over such ontologies, thereby selecting the knowledge relevant to the answer. This method is certainly superior to conventional search for answering queries over well-structured and organized knowledge, e.g., Wikipedia (examples of such queries are Napoleon's year of birth, or city's populations and weather conditions, or the height of mountains in California). Ontological search deeply differs from conventional search in that the work of crawlers is substituted by human-driven knowledge compilation; this is at the same time a virtue and a limitation of the method, as it cannot easily monitor evolving data ontology evolution requires expert work, which currently is provided by humans at limited speed, and will hardly be capable of processing data about, e.g., daily events occurring worldwide. From our vision's perspective, ontology-based search are a new class of search systems, whose expertise is confined within a specific ontological description; they can be considered wider domain-specific systems, but they cannot exhaust the scope of complex search. However, these systems can overcome conventional search engines in their ability of providing answers whose content goes beyond the scope of an individual Web page. [1]

1.3 Building Search Computing Systems

The essence of complex queries is their ability of extracting answers from complex data, rather than from within a single Web page; but complex data require a data integration process. Then, the fundamental question is whether such data integration process can be performed independently – and a-priori – from queries, or should instead be query-specific. In our vision, integration should be query-specific, since answering queries about travels and food or about genes and medical knowledge require intrinsically different data sources: building results for such queries does not require "global data integration", but just data integration relative to specific domains. However, data integration is one of the hardest problems in computing, because it requires full understanding of the semantics of data sources; as such, it cannot be done without human intervention.

With Search Computing, we rely on the work of human experts too, but we move from ontology creation and management, a huge task, to data source

coupling, a somewhat more feasible accomplishment. We denote as data source any data collection accessible on the Web. The Search Computing motto is that each data source should be focused on its single domain of expertise (e.g., travels, music, shows, food, movies, health, genetic diseases), but pairs of data sources which share information (e.g., about locations, people, genes) can be linked to each other, and then complex queries spanning over more than one data source can use such pairing (that we call "composition pattern") to build complex results. An advantage of this approach is its transitivity: if we can pair source A to source B (e.g. pathologies which alter body functions), and then source B to source C (e.g. body functions alterations which are treated by drugs), than we can answer queries that connect A to C (e.g. pathologies treated by drugs) and so on. Each source is responsible of monitoring changes within its domain of expertise, e.g., movie offerings or airfares, through distributed and real-time processing that cannot be performed by knowledge managers, but should remain responsibility of the specialized data sources. Then, the next problem to solve is how to build a composition pattern, i.e., a data source coupling for answering multi-domain queries, recalling that the purpose of composition is search, and that therefore results should be presented to users according to some ranking, respectful of the original rank of the elements coming from the native data sources and of the search intent of the user; indeed, users normally only look at top results of a search, therefore the composition pattern should enable a Search Computing system to produce the highest ranked results first. Our solution is to resort to join, the most popular data management operation, which is however revisited in the context of Search Computing to become service-based and ranking-aware. A result item of a multi-domain query is a "combination", built by joining two or more elements coming from distinct data sources and returned by different search engines; in our first query example ("Where can I attend an interesting scientific conference in my field and at the same time relax on a beautiful beach nearby?"), combinations are triples made of: database conferences (extracted from a site specialized in scholar events, e.g., Dblife), inexpensive flights (extracted from a flight selection site, e.g., Expedia or Edreams), and cities with nice beaches (extracted from tourism or review sites, e.g., Yahoo! Travel or Tripadvisor). Connections carry semantics: flights connect pairs of cities at given dates; therefore connections use "dates" and "cities" as matching properties. We apply joins to the context of software services, by assuming that every data source is wrapped as a web service, and that such services, in most cases, expose a query-like interface which assumes keyword-based input and produces ranked results as output. Services are then

composed by using a ranking-preserving join. We regard such operation as a join of search services.

Search Computing aims at giving to expert users the capability of building similar solutions for different choices of domains, which – in the same way as Expedia or Lastminute – share given properties and therefore can be connected. For such purpose, Search Computing offers a collection of methods and techniques for orchestrating the search engines and building global results. Composition patterns are predefined connections between well identified Web services, therefore orchestrations are not built arbitrarily, but rather by selecting nodes (representing services) and arcs (representing the links in the composition patterns) within a resource network representing the various knowledge sources and their connections. This vision is consistent with the emerging idea of moving from an Internet of (disorganized) pages to an Internet of (semantically coherent) objects. With Search Computing, sources must be registered, and their composition patterns be established. This work requires human intervention, because sources can be linked only by means of join attributes, which must be type-compatible and describe the same real world concept. Source registration occurs by describing the source properties and annotating their role (i.e., representing both input keyword and output result types); when two sources can be joined, a composition pattern is created and associated with a semantic description. This process builds a resource network; we envision communities of users sharing resources in large networks, but also private bodies (e.g., enterprises) developing their own proprietary network of related resources. Then, query processing will use a search computing framework, consisting of a query optimizer – to decide the best order of execution of service calls and the best strategy for joining their results – and an execution engine – monitoring the progressive construction of results and achieving an optimal performance by means of producer-consumer paradigms implementing policies for balancing the frequency of calls to the various services, as well as various levels of caching. The execution engine supports joins of search services as the most relevant operation, and is equipped with mechanisms for regulating join speed to the pace of data production services. We foresee supporting the framework upon general-purpose distributed architectures, such as computing clouds, so as to be easily and effectively available to application providers.

Complex queries are not only hard to answer, but they can be also difficult to formulate for the user. There, an important stream of work is about capturing the user's search intent and directing the user towards the discovery of his

true information need. This process can be done by means of liquid queries, a dynamic query interface that lets users dynamically extend the scope of queries and then browse query results. We expect users to look at results both selectively and globally, possibly asking for more results from given sources, possibly performing grouping and aggregation operations upon result attributes, and so on. The design of the liquid query interface is inspired by Google Squared, whose concept is however extended by the fact that each portion of the result can be traced back to a well-defined data source, thus offering the notion of "data provenance" within complex results. We use a variety of data visualization methods which highlight multiple dimensions and multiple rankings.

The link topology of the resource network also suggests a way to explore the information space by augmenting query results. Complex queries often imply that users have in mind a complex information finding task, which is better represented by an exploration process rather than by a one shot query; the resource network offers a natural way to expand the initial query or its results, by accessing nodes which are reachable from the nodes already used by a query (e.g., expanding the results about action movies by looking at additional information such as its director, actors, and so on, or expanding the notion of geo-localized theatres by looking at public transports or at the nearby pizzerias). Similar capabilities are offered by the latest releases of search systems, such as Bing, which however restricts query and result expansion to domains selected a priori; instead, the resource network could offer query-specific choices.

Finally, we consider the possibility of automatically inferring the relevant network of data sources required to build the answer from keyword based user queries. This will require "understanding" query terms and associating them to resources, through tagging, matching, and clustering techniques; then, the query will be associated to the "best" network of resources according to matching functions, and dynamically evaluated upon them. This goal is rather ambitious, but it is similar to supporting automatic matching of query terms to services within a semantic network of concepts, currently offered by Kosmix. One step in this direction, that we are already considering, is to extend join between services to support the notions of partial linguistic matching between terms (supported by vocabularies such as WordNet) or dealing with the predicate "near" in specific domains (e.g. distance, time, money). [1]

1.4 Building Search Computing Applications

We propose Search Computing as a new method for building a class of rank-aware information finding solutions, accessible to a vast community of Web application developers - and not necessarily confined to large search engine companies. This vision requires a vast community of data providers, who should instrument their data sources so as to become part of broader search environment. Therefore, we are concerned with finding a system of incentives so as to motivate the creation of communities of data providers and of application developers.

The trend towards supporting users in publishing data sources on the Web is a general one. Google, Yahoo and Microsoft are building environments and tools (Fusion Tables, Yahoo! BOSS, and Symphony) for helping Web users to publish their data, with the goal of capturing the so-called "long tail" of data sources. We also consider data publishing essential for Search Computing, however with a specific connotation. Data sources should produce ranked output, organized as lists of items, so that data extraction can be performed incrementally, by "chunks" (sub lists of a given number of results, e.g., 10 items fitting into a page), and users can suspend a search and then resume it, possibly guiding the way in which data sources should be inspected. This data organization, that we call "ranked and chunked", is typically offered by search service APIs (because answering a query normally requires the top few items), but it is not made available by most data sources. However, most data sources can be turned into "ranked and chunked". Ranked data extraction is currently supported by query languages, and chunking can also be programmed on top of tabular data representations, by using top-k extraction commands. Such provisions apply to data which are initially materialized and mapped into suitable formats. Therefore, we are building tools and/or providing best practices, applicable to data sources of various kinds, for enabling data providers to build "search" service adapters. We design methods and tools which will take into account the most popular data publishing environments, provided by the major players in the field (examples are Yahoo! Search Boss and Google's Fusion tables), so as to maximally ease the task of writing adapters.

The "vision" of Search Computing builds upon two new communities of users:

- Content providers, who want to organize their content (now in the format of data collections, databases, web pages) in order to make it available for search access by third parties. They will be assisted by the availability of a deployment environment facilitating at most their task,

and will be provided with the possibility to register their data within a community. In this way, the "long tail" of content providers will see a concrete possibility of exploitation.

- Application developers and/or expert users, who want to offer new services built by composing domain-specific content in order to go "beyond" general purpose search engines such as Google and the other main players. They will be assisted as well by the availability of visual tools facilitating at most their task, and will in addition find a deployment environment, either obtained by installing run-time components upon their servers, or - most interestingly - by finding servers already deployed within cloud computing architectures, where they will run their applications.

In the simplest scenario, the same person or organization may play the role of content and application provider, and offer to generic users the access to a specific content. In the most interesting and challenging scenario, application developers would act as the brokers of new search applications, built by assembling arbitrary resources, accessible through uniform service interfaces; some of them could be generic, world-wide, and powerful (e.g., general purpose search engines or geo-localization services), other resources could be specialized, local, and sophisticated (e.g., the “gourmet suggestions” about slow-food offers in given geographic regions). Moreover, expert users might visually compose queries, starting directly from resource networks, thus covering the gap in expressing a complex semantics to new generation search engine.

Most of the effort in Search Computing will then be dedicated to supporting content providers, application developers, expert users, and end users. We expect application developers to be aware of the resource networks and use visual tools for building applications with a high-level approach, consisting in using visual tools for selecting resource sub-networks and turn them into parametric query templates. The boundary between such actors is not completely sharp, as we do expect some users to be expert to the point of setting up an application themselves. In this vision, new business options open up for service providers and brokers, with appropriate licensing agreements regulating the rights to content access and the sharing of profits based upon accountability of the click-through generated traffic or of actual committed transactions. This vision is compatible with the current models adopted by the major search engine companies (e.g., Google or Yahoo), which monitor the

click-through traffic generated by advertising and sponsored links. Some of the aspects considered in this research plan can be considered futuristic and difficult to accomplish, but Search Computing is a five-year project. This book reports the results of the first year of the project and illustrates our first moves to accomplish our vision. Four more years of investigation and development are ahead of us; the results of the project are available (now and throughout the project) on the project's website. [1]

1.5 Contribution of the thesis

The aim of this thesis is to analyze, evaluate, enhance, and implement the existing Information Retrieval diversification approaches described in the paper [2] .My original contribution can be summarized as follows:

1. We introduce and implement the concept of semantic diversification rules in order to do diversification based on domain specific criteria. For example, we have implemented Normal distance and Geographical distance. Also, we handle the semantic diversity functions that require the prior computation of derived attributes, like Euclidean distance.

2. We enhance and give a better version to get normalize distance by considering normalization on each quantitative semantic rule and also consider their maximum and minimum score value.

3. We introduce and implement a new kind of diversity, which is the combination of existing diversification types, like Object equality (Categorical diversification) and Attribute distances (Quantitative diversification).

4. We formalize and implement the weight-based semantic rules diversification, where the diversification criterion can be a linear combination of different sub-criteria so that one semantic rule can be declared to be more important than the other.

5. We implement the relevance and diversification algorithms (MaxSum, MaxMin, and MMR) in the pipeline of the SeCo architecture using SeCo manipulation functions.

6. We analysis the complexity of the diversification score functions, MaxSum, MaxMin, and MMR.

7. We implement the recall and precision evaluation based on SeCo data services.

8. We have made the implementation configurable based on the configuration of the SeCo data access function.

9. We evaluate the existing SeCo functions and give valuable comments on their enhancement, like handling of JSON inside the Manipulation Function's configuration.

Chapter 2. Problem Statement

Multi-domain search tries to respond to queries that involve multiple correlated concepts, like: “Find an affordable house in a city with low criminality index, good schools and medical services" or “Find an ICT conference in a venue with a close-by beach, affordable accommodation and good restaurants within walking distance". The information to answer these queries is available on the Web, but deciding on the best global option requires gluing together several scattered pieces of information, not an easy task for common Web users. Multi-domain search has the potential of bridging the gap between general purpose search engines, which are able to retrieve instances of at most one entity at a time (e.g., cities, products), and vertical search applications in specific domains (e.g., trip planning, real estate), which can correlate only a fixed set of information sources. The increasing availability of Web-accessible data sources, including search engine APIs, products, events and people databases (e.g., Amazon, Eventful, LinkedIn), scientific data sources (e.g., DBPL, PubMed), community curated data sources (e.g., YQL Open Data Tables, DBpedia) constitutes a formidable platform for novel search applications supporting complex information seeking processes that span multiple domains. Formally, multi-domain queries can be represented as rankjoin queries over a set of relations, representing the wrapped data sources [3] [4]. Each item in the result set is a combination of objects that satisfy the join and selection conditions, and the result set is ranked according to a scoring function, which can be expressed as a combination of local relevance criteria formulated on objects or associations (e.g., price or rating for a hotel, distance between the conference venue, hotel, and restaurant). Due to the combinatorial nature of multi-domain search, the number of combinations in the result set is normally very high, and strongly relevant objects tend to combine repeatedly with many other concepts, requiring the user to scroll down the list of results deeply to see alternative, maybe only slightly less relevant, objects.

As a running example, consider multi-domain search scenarios where three data sources are wrapped by the following relations:

Hotel (**HName, HLoc, HRating, HPrice**)
Restaurant (**RName, RLoc, RRating, RPrice**)
Museum (**MName, MLoc, MRating, MPrice**)

Each entity is described by its name (key attribute), geo referenced location, users' rating, and price. A query issued with a mobile phone over these data sources aims at finding combinations within a short distance from the user location and good ratings, to be returned in order of total price. If we suppose to find 100 hotels and restaurants and 20 events, and assume a 10% selectivity of the join and selection condition

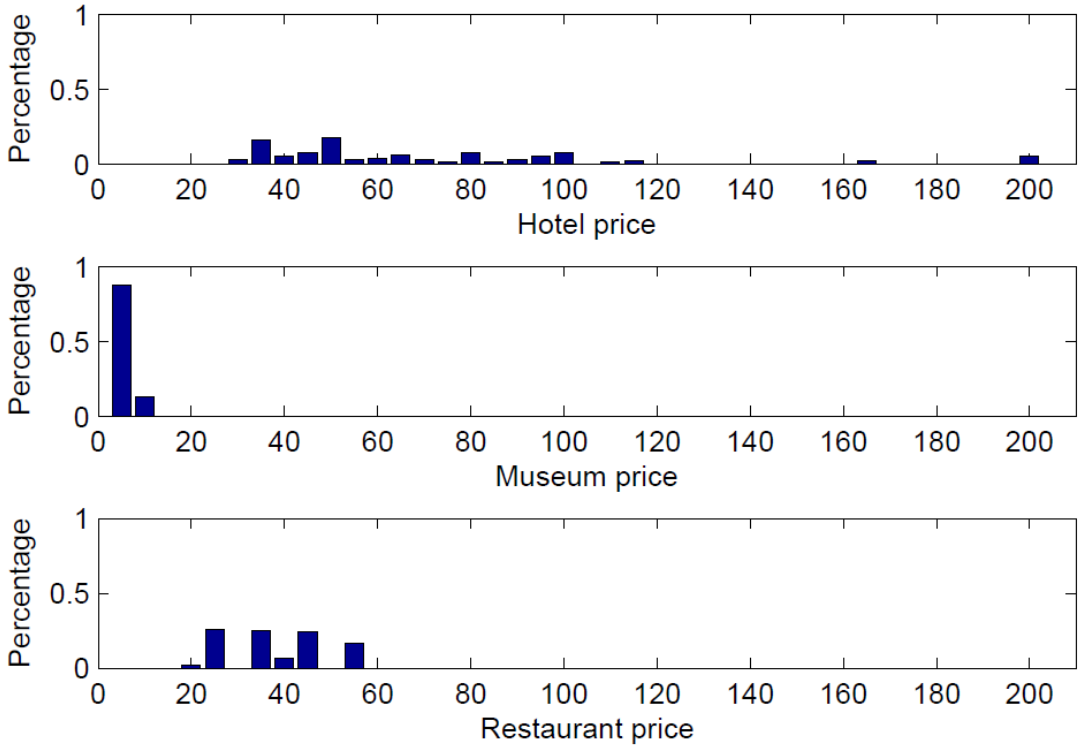


Figure 2-1 Score distribution of Table 2-1 result set

on distance and minimum rating, a total number of 20000 combinations can be used to build the top-K result set. Supposing to show only 10 combinations disregarding the relevance of the constituent objects, up to 30 distinct objects out of 220 can be presented (14%). However, in real situations, the composition of the top-K results also depends on relevance, which decreases diversity. For illustration, Table 2-1 shows a result set, which contains the top-10 combinations ranked according to total price. We observe that the result is rather poor in terms of diversity, as only 1 hotel, 3 restaurants and 4 museums are represented. Indeed, the number of distinct objects that appear in the top-K results is sensitive to the distribution of attribute values used to compute the score of the combination. In our case, the price range of good-rated hotels is larger than the price ranges of restaurants and events, as illustrated in Figure 2-1. Hence, budget hotels will appear repeatedly in the top-K list, lowering the number of distinct objects seen by the user. The same observation applies when, fixed a hotel, one considers the price range of restaurants compared to the price range of museums. Improving the diversity of the result set is the aim

of diversification, which can be defined in the context of multi domain search as the selection of K elements out of a universe of N combinations, so to maximize a quality criterion that combines the relevance and the diversity of the objects of distinct types seen by the user. In this respect, Table 2-2 shows an example of result set with diversified combinations. We observe that the set does not necessarily contain the top-10 combinations in terms of total price. Nevertheless, the result is much richer: 3 hotels, 5 restaurants and 7 museums are selected. The thesis is summarized as follows:

- We introduce the problem of diversification in the context of multi-domain search, an area made quite interesting by the increasing availability of public "joinable" Web data sources.

- We formalize multi-domain diversification following the approach in [5], and propose three criteria for comparing combinations (categorical and quantitative diversity, mix diversity).

- Since diversification can be shown to be NP-hard also in the multi-domain context, we study the behavior of three known greedy algorithms (MinMax, MMR, MaxSum) experimentally. Specifically, we test the hypothesis that the diversification algorithms improve the quality of the result set with respect to a baseline constituted by the selection of the most relevant K combinations; we compare the degree of diversification induced by the three algorithms, when using both categorical and quantitative diversity, according to well established objective measures tailored to the context of multi-domain search.

- We formally analyze under which conditions diversification can be potentially effective in improving the quality of the results. In particular, we consider the impact of the score distributions on the diversity of the result set which includes the top-K combinations based on relevance only.

- We evaluate the perception and utility of diversification in multi-domain search with user study based on selection tasks involving combinations of correlated objects: focuses on explicit comparison of result sets diversified according to different algorithms [2]

- We introduce and implement a new concept of handling complex diversification attribute like Geo Graphical distance, Euclidean distance.

- We implement the real time application for calculating the diversity score based on categorical, quantitative diversity, mixture of categorical and quantitative diversification with their semantic meanings. Mix diversification type is a new concept which did not existed before.

- We implement the weight based semantic rule to enable favoritism between semantic rules (previously, knows as attribute). For example, user want diversification based on the Hotel price and Restaurant price, but he is more concern about the Hotel price than the Restaurant price. Favoritism is also a new contribution of this thesis.

- Previously, we were doing normalization based on the complete score for quantitative diversification, which creates problem when we have more than one quantitative diversification type where one attribute distance is much greater than the other, and also we did not check minimum distance value. But now in this thesis, we handled this problem of normalization by doing the normalization on each and every quantitative semantic rule consider the maximum and minimum rule score.

- Based on the diversification score implementation we did a real implementation of relevance and diversification algorithm (MaxSum, MaxMin, and MMR) which is fully plug and playable with SeCo architecture.

- We implement the method to calculate novelty-relevance, and recall measurement. Though, the graphs from those values and generated using Microsoft Excel.

	Hotel		Restaurant		Museum		$S^{(a)}(\tau, q)$
	HName	**H Price**	**RName**	**R Price**	**MName**	**M Price**	**Total price**
τ_1	Hotel Amadeus	€35	Miyako	€25	Galleria d'Arte Moderna	€0	€60
τ_2	Hotel Amadeus	€35	Miyako	€25	Museo Civico di Milano	€0	€60
τ_3	Hotel Amadeus	€35	Miyako	€25	Museo di Storia Contemporanea	€0	€60
τ_4	Hotel Amadeus	€35	Porca Vacca	€25	Galleria d'Arte Moderna	€0	€60
τ_5	Hotel Amadeus	€35	Porca Vacca	€25	Museo Civico di Milano	€0	€60
τ_6	Hotel Amadeus	€35	Porca Vacca	€25	Orto Botanico di Brera	€0	€60
τ_7	Hotel Amadeus	€35	Spontini 6	€25	Galleria d'Arte Moderna	€0	€60
τ_8	Hotel Amadeus	€35	Spontini 6	€25	Museo Civico di Milano	€0	€60
τ_9	Hotel Amadeus	€35	Spontini 6	€25	Orto Botanico di Brera	€0	€60
τ_{10}	Hotel Amadeus	€35	Spontini 6	€25	Museo di Storia Contemporanea	€0	€60

Table 2-1 Un-normalized Result

	Hotel		Restaurant		Museum		$S^{(a)}(\tau, q)$
	HName	**H Price**	**RName**	**R Price**	**MName**	**M Price**	**Total price**
τ_1	Hotel Amadeus	€35	Miyako	€25	Galleria d'Arte Moderna	€0	€60
τ_2	Hotel Amadeus	€35	Porca Vacca	€25	Museo Civico di Milano	€0	€60
τ_3	Hotel Amadeus	€35	Miyako	€25	Orto Botanico di Brera	€0	€60
τ_4	Hotel Delle Nazioni	€36	Miyako	€25	Galleria d'Arte Moderna	€0	€61
τ_5	Hotel Delle Nazioni	€36	The Dhaba	€25	Orto Botanico di Brera	€0	€61
τ_6	Hotel Delle Nazioni	€36	Spontini 6	€25	Pad. d'Arte Contemporanea	€2	€63
τ_7	Hotel Zefiro	€39	Matto di Bacco	€25	Galleria d'Arte Moderna	€0	€64
τ_8	Hotel Zefiro	€39	Porca Vacca	€25	Museo Civico di Milano	€0	€64
τ_9	Hotel Delle Nazioni	€36	Porca Vacca	€25	Museo della Permanente	€6	€67
τ_{10}	Hotel Zefiro	€39	Miyako	€25	Museo Civico di Storia Naturale	€3	€67

Table 2-2 Normalized Result

Chapter 3. Related work on diversification

The focus of this thesis is the evaluation of how diversification can improve results presentation for multi-domain search applications applied to Web data sources. Two areas of related work are prominent: the evolution of search systems towards the extraction of structured information from Web content and the diversification of results for document collections and structured data.

3.1 Concept Search and Multi-domain Search

Concept search [6], i.e., the task of extracting entity instances from collections of documents, has reached enough maturity to be incorporated in mainstream search engines, which now can recognize concepts like cities, people, and products and build responses that include structured information (e.g., maps, news, and weather reports for a city). Furthermore, topical search engines, like Kosmix [7], map keyword queries to a large taxonomy of topics and display result pages assembled from a variety of data sources. Mining concept structure is also the focus of [8], which concentrates on the extraction and presentation of composite concepts, defined as objects composed of a central item and several sub-objects, called packages, respecting a global constraint on a join attribute, and applying a visual effect optimization. Multi-domain queries extend concept queries and composite object construction to the case of multiple entities and association paths of arbitrary length. Multidomain search focuses on processing queries involving several topics and domains on Web data sources. It comprises techniques for: discovering and registering the data sources, which can be APIs over the deep Web, linked data repositories, or wrapped Web sites; performing efficient rank-join queries on the selected data sources; merging local ranks (e.g., hotel stars, home prices) into a global rank and presenting results (i.e., combinations of correlated objects) in an interface that allows provenance tracking, query refinement, and data space navigation. The present work explores diversification in the context of multi-domain search, as a mean for improving the utility of result sets made of associated entity instances.

3.2 Diversification

Result diversification is a well-investigated topic: recent works overview and classify the existing approaches [9] [10] and provide a systematic axiomatization of the problem [5], which is the base of the formalization of multi-domain diversification. A broad distinction can be done between the

contributions that focus on diversifying search results for document collections [11] [12] [13] and those that concentrate instead on structured data sets [14] [15] [16] [17].

Our work is mostly related to diversification applied to structured data. In this field, the work in [17] examines the diversification of structured results sets as produced by queries in online shopping applications. The thesis shows how to solve exactly the problem of picking K out of N products so to minimize an attribute-based notion of similarity and discusses an efficient implementation technique based on tree traversal. Relevance is also considered, by extending the differentiation to scored tuples: the choice of K items minimizes the measure of similarity but at the same time produces only K-subsets with maximal score (in some sense, giving precedence to relevance w.r.t. diversity). The base of the proposal is the availability of a domain dependent ordering of attributes (e.g., car's make, model, year, color, etc), which is used to build a prefix-tree index supporting the selection of the subset of objects with maximal score end minimal similarity. Multi-domain diversification, as discussed in this thesis, is a broader problem; it could be partially reduced to prefix-based diversification only in the case of categorical diversity, by choosing an arbitrary order for the categorical attributes used to measure combination diversity. Keyword search in structured databases is addressed in [14], where diversification is not applied to result sets, but to query interpretations, which are assumed to be available from the knowledge of the database content and query logs. The multi-domain search applications addressed in this thesis assume for simplicity unambiguous queries and thus a fixed interpretation, but could reuse the interpretation diversification approach of [14] to cope for multi-domain searches with more than one possible interpretation. A very recent related work is [16], which applies to the selection of Web services characterized by their non-functional properties. MaxMin might overemphasize outliers and address this issue by introducing a novel diversification objective, MaxCov:

$$F(\mathcal{R}_K) \;\; = - \max_{\tau \in \mathcal{R} \setminus \mathcal{R}_K} \left\{ \left(S(\tau, q)^{\lambda} \right) \min_{\acute{t} \, \epsilon \, \mathcal{R}_k} (\tau, \acute{t}) \right\}$$

which leads to the selection of items with high relevance score, such that the remaining ones are not too far from any of the elements of the diversified result set. Solving the problem is NP-hard and [16] contains a 2-approximation algorithm, which could be applied to multi-domain search, simply by replacing a line in MaxMin with:

$$\tau^* = -\underset{\tau \in \mathcal{R} \setminus \mathcal{R}_K}{\operatorname{argmax}} \left\{ \left(S(\tau, q)^{\lambda} \right) \min_{\hat{t} \in \mathcal{R}_k} (\tau, \hat{t}) \right\}$$

and initializing the result set with the most relevant combination. We plan the testing of MaxCov as part of the future work. Finally, the work [15] investigates the diversification of structured data from a different perspective: the selection of a limited number of features that can maximally highlight the differences among multiple result sets. Although the problem is apparently different from multi-domain search (the actual goal of [15] is to find a set of attribute values that maximally differentiates a number of input results set, respecting a size upper bound) identifying the best attributes to use for ranking and diversification is relevant to multidomain search as well, and we have started addressing it by studying how the distribution of attribute values affects the capability of the ranking function to sample the population of the input relations evenly. A complementary approach for result diversification to solve the issue of too many answers in structured data queries is based on clustering. For example, the work in [18] automatically constructs a small set of representative tuples that summarize the (long) result set of a database query. Good representatives are determined using k-medoids clustering, which merged as the most effective summarization approach in a user study. Although the work in [18] considers only one entity, it would en intersecting development to extend its approach to result summarization to entity combinations and contrast the effectiveness of clustering and diversification in a broader user study.

Chapter 4. Architecture Design

SeCo Platform consists of the following modules:

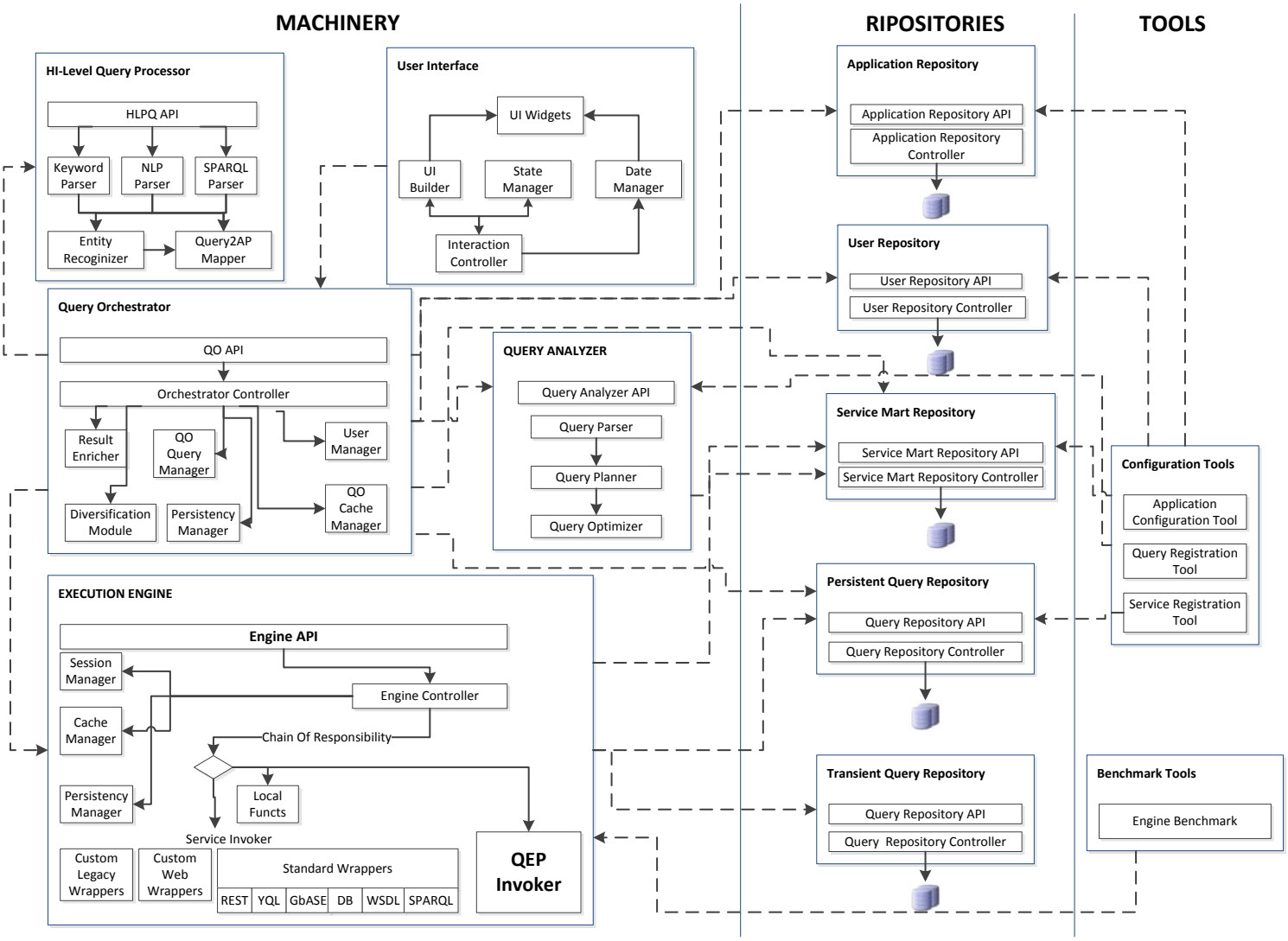


Figure 4-1 SeCo Architecture

1. **[UI] User Interface:** it's front-end of a SeCo system devoted to the user interaction with the system; it is dynamically configured according to the executed application. It accepts an input user commands and produces queries and query expansion commands for the SeCo query orchestrator.

2. **[QO] query orchestrator:** it manages user sessions; it keeps track of the current interaction status and history; it orchestrates the invocation of

the SeCo execution engine and of the high level query processor, according to the format of the query; it interacts with the query analyzer to create executable query plans; it acts as a proxy toward the service mart repository, so to allow end-users to explore the set of services available for the current SeCo system; it interacts with the query repository to retrieve the list of one-shot query plans saved in a SeCo instance; it interacts with the user repository to access information about the users of one or more SeCo Application; it interacts with the application repository to retrieve the application configuration files.

3. **[HQ] high level query processor:** it accepts as input an unstructured user query (e.g., a set of keywords, a natural language sentence, or a structured English sentence) or a declarative query expressed over an ontology (or, more in general, a schema), producing as output the query expressed in the SeCoQL.

4. **[EE] execution engine:** it takes in input a reference to a physical plan and executes it, by driving the invocation of the needed services. Depending on the format of the query, it may interact with the query analyzer (for query planning and optimization). It contains structures for session level caching of results. The engine offers built-in wrappers for the invocation of several Web based infrastructures (e.g., YQL, GBASE), query end-point (e.g. SPARQL) and resources (e.g., WSDL- and REST-based Web services). It also supports the invocation of legacy and local data sources. The execution engine interacts with the service mart repository to retrieve the concrete description of the search services invoked in a physical plan.

5. **[QA] query analyzer:** Query analyzer takes in input a query in SeCoQL, parses it, and translates it into a logical plan (according to some heuristics) and, then into a (possibly optimized) physical plan.

6. **[SM] service mart repository:** Service mart repository registers data sources at multiple level of abstraction: Service Marts, Access Patterns and Connection Patterns, and Service Interfaces. It provides a view on the Service Interfaces suitable for consumption by the execution engine.

7. **[PQR] persistent query repository:** it registers queries at multiple level of abstraction: SeCoQL, logical plan, and physical plan.

8. **[TQR] transient query repository:** it registers queries at multiple level of abstraction: SeCoQL, Logical Plan, and Physical Plan. The lifetime of such queries is bounded to an engine execution session.

9. **[UR] user repository:** it registers the users of one or more SeCo Applications.

10. **[AP] application repository:** it registers the configuration files (query configuration, result configuration, etc.) required by the user interface for a SeCo Application.

11. **[CT] configuration tools:** set of design and configuration tools that enable designers and administrators to create and manage the services, queries, and applications currently registered in a SeCo System through a visual, web-based interface.

12. **[BT] benchmark tools:** a set of monitoring and probing tools able to mimic and inspect the behavior of the SeCo system.

4.1 Query Management in SeCo

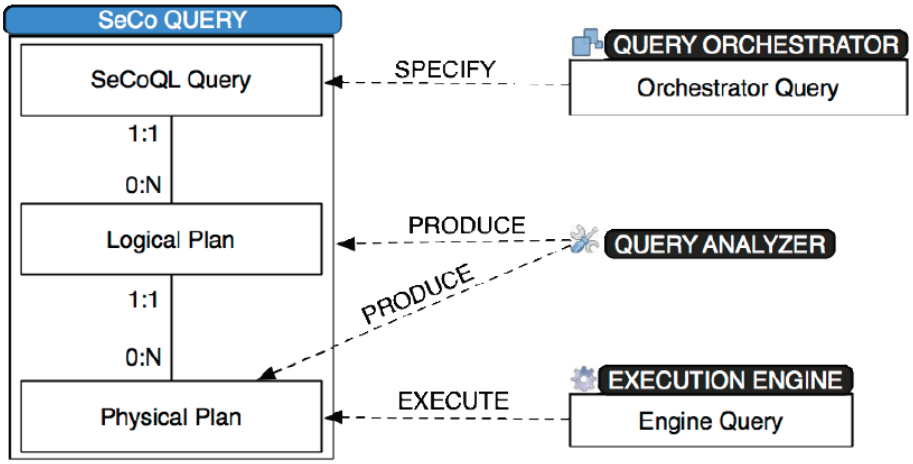


Figure 4-2 Structure of a SeCo Query and relationship with the architecture components

In SeCo, queries are first-class citizens that are created, translated, manipulated and consumed by the system's components.
A **SeCo query** is composed by:

1. **SeCoQL query:** a declarative representation of a SeCo Query. A SeCoQL Query is expressed in a SQL like query language defined over access patterns A SeCoQL query has two isomorphic representations:
    - A declarative representation, expressed as described in [2], for which a dedicated parser is required

```
DEFINE QUERY NightPlan($X:String, $Y: Integer , $U:String, $V:String, $W:String) AS
        SELECT M.*, T.*, R.*, TotalPrice=T.Price + R.AvgPrice
        FROM ((Movie (iGenre: $X, iYear: $Y) AS M USING IMDB_MOVIES,
        JOIN
        Theatre (iAddress: $U, iCity: $V, iCountry: $W) AS T USING
        GOOGLE_DISPLAYING
                ON M.Title=T.Title)
                JOIN Restaurant (iCountry: $W, iCategory: "Italian Restaurant") AS R USING
        YQL_LOCAL
        ON T.address=R.Address AND T.city=R.City)
        WHERE R.AvgPrice<30 AND TotalPrice<40
        RANK BY (R=0.4, T=0.3, M=0.3)
        LIMIT 20 TUPLES AND 50 CALLS
```

Figure 4-3 Declarative SeCoQL query Example

2. A serialized representation, directly consumable by the query analyzer
3. **Logical Plan:** a logical plan is the product of the planning activity performed by the *query analyzer* on a **SeCoQL** query. It defines, at a logical level, the order of execution and the dependencies among services in a query. In the current version of the system, the query analyzer can produce one Logical Plan out of a **SeCoQL** query, implementing the heuristic “parallel is better”.

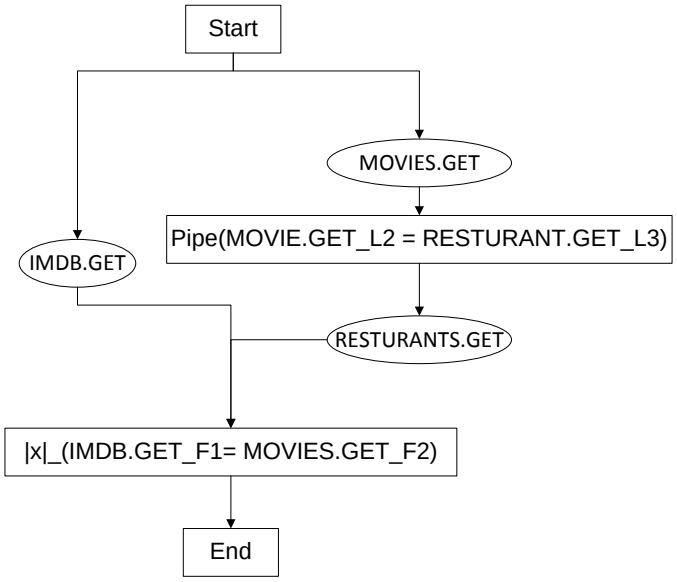


Figure 4-4 Logical plan example

4. **Physical Plan:** a physic plan is the product of the translation activity performed by the query analyzer over a logical plan to create a query representation executable by the execution engine. The query analyzer can produce one Physical Plan out of a Logical Plan.

5. **Engine Query:** an engine query is an execution of a given physical plan (related to a single given logical plan) performed by the execution engine. An engine query, hence, is fed with all the input parameters required for the execution of the physical plan. Figure 5 depicts an example of Engine Query.

```
{
        "sourceId": "run_ex",
        "accessMethodId": "outer_pipe",
        "inputTuple": {
                "address": "Time Square",
                "city": "New York",
                "country": "USA",
                "genre": "Drama",
                "year": "2009",
                "category" : "Meat restaurant"
        }
}
```

Figure 4-5 Query Engine Example (Add/Update Conditions on attribute)

An **engine query** is identified by two attributes:

- **sourceID:** the identifier of the query source in a query repository.
- **accessMethodId:** the identifier of a SeCo Query physical plan within a query source.

6. **Orchestrator Query:** an Orchestrator query is a query addressed to the query orchestrator to perform a SeCoQL query on the SeCo system [5]. Given that an Orchestrator Query can be an exploratory query, it can contain reference to the execution of several engine queries. An Orchestrator query may be initially expressed as a SeCoQL query or as a reference to a Logical or Physical Plan.

```
{
	"Query": {
		"type": "newQuery",
		"sessionID": "$sessionID",
		"queryID": "$queryID",
		"sourceType": "query",
		"sourceID": "qep_movies",
		"accessMethodID": "outer_pipe",
		"UserInput": [{
				"inputID": "address",
				"inputValue": "Time Square"
			},{
				"inputID": "city",
				"inputValue": "New York"
			},{
				"inputID": "country",
				"inputValue": "USA"
			},{
				"inputID": "genre",
				"inputValue": "Drama"
			},{
				"inputID": "year",
				"inputValue": "2009"
			}]
	},
	"resultFormat": {
		"serviceAlias": "2",
		"onlyJoin": "true"
	}
}
```

Figure 4-6 An example of Orchestrator Query

4.2 SeCo Query Life Cycle

The life cycle of a query in SeCo develops as follows:

1. A SeCoQL query is created manually by an expert user or automatically by a System's component (e.g., the query orchestrator or the query registration tool); the SeCoQL query can be:
    - A **one shot query**: the query does not refer to any query defined in the current user session.

- An **exploratory query**: the query refers to a query defined in the current user session, expanding it with an additional service or query.

2. The SeCoQL query is stored as a new physical query plan in a query repository (persistent or transient). A physical query plan is logically contained in a physical query source, i.e. a logical container for a set of related SeCo queries.

3. The SeCoQL query is provided to the query analyzer, which, in turn, parses it, creates one (or more) Logical and Physical Plan. The invocation of the query analyzer is performed:

   - By the execution engine, as a result of a new engine query instantiation.

   - By a third party component (e.g., the query registration tool).

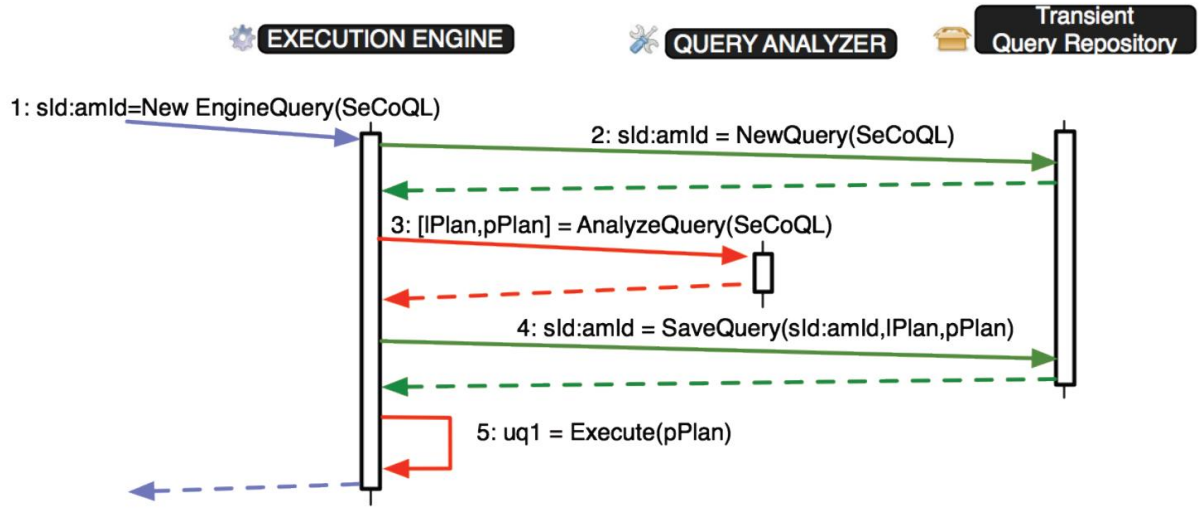


Figure 4-7 Exploratory query creation and execution

The Logical Plan-Physical Plan couple is stored in the query repository, and it is associated with the same query source as the original SeCoQL query. If the SeCoQL query has been created by means of the query orchestrator (e.g., as the result of a query expansion on a exploratory query), then the physical plan is stored in the transient query repository, to be purged when the query session expires.

Notice that the current implementation does not enforce SeCoQL, Logical Plan, or Physical Plan caching thus allowing the existence of duplicated SeCoQL queries and plans in the query repository.

4.3 SeCo Queries and user interaction

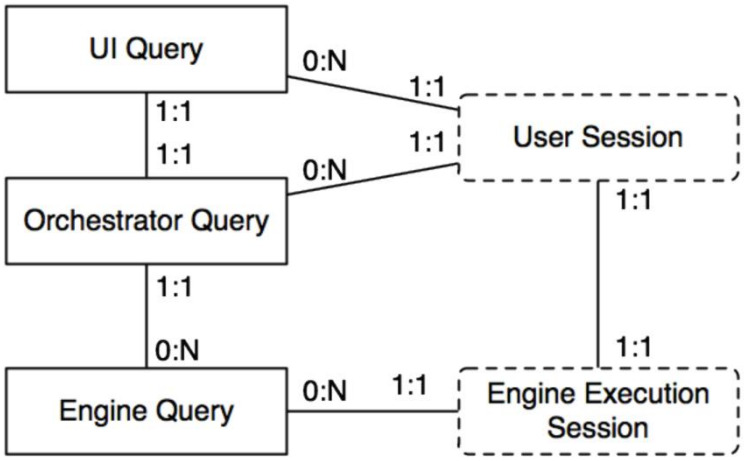


Figure 4-8 Relationships among queries and sessions

For what concerns the user experience, a query life cycle develops as follows:

- The user initiates a new user session by connecting to the user interface.
- The user creates a new UI Query, by:
  - Selecting an existing one-shot query from the persistent query repository.
  - Selecting an access pattern from the service mart repository.
- A new Orchestrator Query is created.
- If not already existing, the query orchestrator starts a new engine execution session with the execution engine.
- The query orchestrator creates a new engine query providing:
  - a SeCoQL query OR a the identifier of a physical query plan (contained in a physical query source) that implements the SeCo query
  - a sets of input parameters
- The execution engine retrieves the query descriptor of the physical plan to execute from the query repository.
- The *query* orchestrator fetches the results from the engine and provides it back to the user interface.
- The user can now perform local or remote interactions on the result set.

- Local interactions do not involve the query orchestrator; therefore, the state of the current orchestrator query does not change; examples or local interactions are local sorting of results, local filtering, clustering, etc.
- Remote interactions, instead, affects the state of both orchestrator and engine queries, as they require a new interaction with the query orchestrator and with the execution engine. Examples of remote interactions are: remote filtering and sorting, requests for more results.

If the current UI Query is an exploratory query, then the user can perform an expand operation. The user initiates such operation when he/she selects a new access pattern from the service mart repository. When a user performs an expansion:

- The state of the current UI and orchestrator query are modified to include the new information.
- The query orchestrator creates a new engine query providing a new SeCoQL query and a set of input parameters. Notice that the previous engine query is not closed, as it might be re-used during the exploration process.
- The execution engine retrieves the query descriptor of the physical plan to execute from the query repository, and the execution proceeds as before.
- The user can always create a new UI Query as before. Notice that the old UI Query (and, consequently, the old orchestrator query) remains active, thus amenable for further user interactions.

4.4 SeCo Query History

The system is able to keep track and maintain several UI queries and Orchestrator queries for a given user session. Therefore, each UI/Orchestrator query is associated with a query history.

The state of query history is made of a set of user interactions on the system. An interaction can be performed locally or remotely, w.r.t. the end-user (i.e. the user interface).

Examples of remote operations are:

1. A filter/sort operation, performed by the execution engine, on the results of the current engine query**;**
2. A change in the ranking function of the current engine query;

3. The creation of a new engine query as the result of the definition of a new SeCoQL query. A new SeCoQL query can be created:
    a. Because the user changed/added/removed a constrain in the query (e.g., the user wants to add an additional filter to the previous SeCoQL query);
    b. The user performs an exploratory step

Remote operations always involve Orchestrator Queries; therefore the query orchestrator handles their history.
Notice that the first two operations can be applied to both one shot and exploratory queries, while the fourth one is available for exploratory queries only.
Examples of local operations are all the ones that produce a change in the visual appearance of the result in the user interface, i.e. the application of a local query command like:

1. local sort
2. local filter
3. grouping
4. clustering
5. etc.

Local operations just involve UI queries; therefore their history is handled by the user interface. Notice that changes in the currently selected result visualization (e.g., tabular, atom view, map, etc.) are not considered as a local user interaction meaningful for query history purposes.

4.5 Manipulation Function

A manipulation Function (MF) is a function that modifies the response of a SeCo interaction. Diversification is one kind of manipulation function. There are 3 kind of MF depending on what they change:

4.5.1 Schema MF

Schema manipulation function changes the schema of results by adding some attribute to a service output. Even if the Orchestrator allows any changes on the Schema, you are not allowed to remove or change the type of existing fields. Removing fields or changing their type can cause errors on future expansions that could involve those fields.

4.5.2 Instance MF

Instance MF changes the results tuples by for example, removing some tuples, changing the score, changing a field value etc. No type check is performed on the manipulated tuples; it's up to the MF to be sure to fulfill the data type constraints (assign a numeric value to an integer field and so on).

4.5.3 Properties MF:

Properties manipulation function is used to add properties in the results set.

4.6 Diversification Architecture

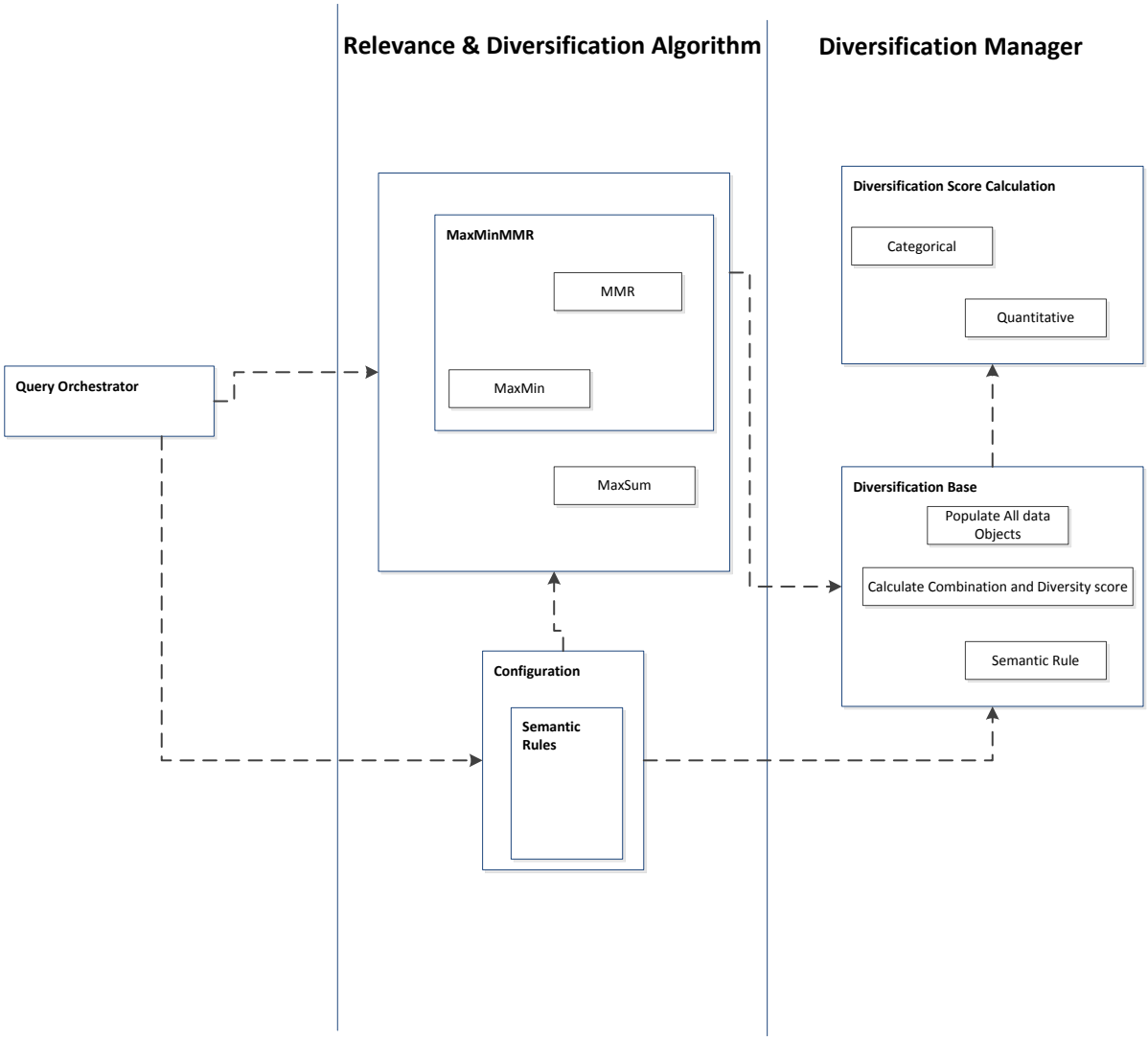


Figure 4-9 Diversification Architecture

4.6.1 Configuration

In order to use a pipeline (a list of ordered MFs that have to be called in succession) in the orchestrator a configuration must be written in the configuration file of Orchestrator. For each diversification defined in the file:

- **ID:** univocally identifier of the pipeline
- **Enable:** (true or false): if true the pipeline is executed during the execution of the other interactions (newQuery, Expand, Filter etc), else it can be executed only as a service/interaction.
- **Goals:** Comma-separated list of interaction (FirstQuery, Expand, ReRank, Filter, Sort) after which the ORCH invoke the pipeline (ignored in case of execution as a service/interaction).
- **InvolvedServices**: groups of serviceInterfaceId that involve the pipeline, the pipe are invoked only if in the current plan there are all the services in at least one of the groups. (e.g. a pipe can be invoked if the execution plan contains an invocation to the Cinema service or both restaurant and hotel); null if the pipeline has to be call with every service.
- **Mfs**: list of MF:
  - **Id**: univocally identifier of the MF
  - **Type**: one of schema, instance, properties
  - **className:** Qualified Class Name
  - **folder:** name of the folder that contains the project jar
  - **method:** name of the method that performs the MF
  - **configuration:** a list of name and JSON property value used to configure the MF.

```
{
   "pipelines": {
        "HRM_Price_Example":{
        "enable": "true",
        "goals": "all, filter, expand, new",
        "involved_services": [ ["9Mip3B1IpOnFHJPZNJD3CH"]],
        "mfs": [
          {
            "id": "5",
            "type": "schema, instance, properites",
            "className": "org.seco.diversification.MaxSum",
            "folder": "diversification",
            "method": "invoke",
            "configuration": []
          }
        ]
      }
   }
}
```

Figure 4-10 Diversification Configuration

- **diversityCoefficient:** describe how much diversification we need in the result set. It has maximum value is one and minimum value is zero.
- **topKTuple** describe how many tuple we need in the result set. If the total number in the provided data is less than "topKTuple" than system return all available data tuples.
- **SemanticRules** define the meaning based on which we need the diversification.

Following configuration is required for every diversification algorithms.

```
"configuration": [
          {
                    "name": "diversityCoefficient",
                    "value":  importance of the diversification
          },{
                    "name": "topKTuple",
                    "value": number of tuple that we want in the result set
          },{
                    "name": "SemanticRules",
                    "value":  Json of the Semantic Rule
          }
]
```

Figure 4-11 Template for diversification configuration

4.6.1.1 SemanticRules

Semantic rule is a composition of the following attributes:

- **semanticType** can be “SimpleDiff” like 3-2=1 or it can be some complex type like “CoordLong” (for calculating the geographical difference between two points), “EuclDis”(for calculating the Euclidean distance).

- **attributeName** is the list of attribute in the data schema that is needed to calculate “semanticType”.

- **diversificationWeight** is the relevance weight of one rule, ranging from zero to one. Total weight of all the rules should be one.

- **schemaId** is the data schema Id where we need to find out the “attributeName” list. It’s required because it may be more than one schemas has the same attribute list that is present in the semantic rule.

- **diversificationType** can be "categorical" or “quantitative” based on the nature of the attribute.

A complete example of the configuration is shown on the next page.

```
{
   "pipelines": {
        "HRM_Price_Example":{
         "enable": "true",
         "goals": "all, filter, expand, new",
"involved_services":[["9Mip3B1IpOnFHJPZNJD3CH","GjvxAw2xhlnTPEx33A6SeK","1X9oodXQQKlahX2pYO8P7g
" ]],
         "mfs": [ {   "id": "5",
              "type": "schema, instance, properites",
              "className": "org.seco.diversification.MaxSum",
              "folder": "diversification",
              "method": "invoke",
              "configuration": [ {  "name": "diversityCoefficient",
                   "value": 0.9
                 },{ "name": "topKtuple",
                   "value": 100
                 },{ "name": "enableDataSorting",
                   "value": true
                 },{"name": "requestSavePath",
                   "value": "C:/logDiver/serializedObj/"
                 },{ "name": "SemanticRules",
                   "value": [ {"semanticType": "SimpleDiff",
                                                    "attributeName":["lowestPrice"],
                                                    "diversificationWeight":0.33,
                                                    "schemaId": "9Mip3B1IpOnFHJPZNJD3CH.1",
                                                    "diversificationType":"quantitative"
                                               },{"semanticType": "SimpleDiff",
                                                    "attributeName":["avgPrice"],
                                                    "diversificationWeight":0.33,
                                                    "schemaId": "GjvxAw2xhlnTPEx33A6SeK.2",
                                                    "diversificationType":"quantitative"
                                               },{"semanticType": "SimpleDiff",
                                                    "attributeName":["fullFee"],
                                                    "diversificationWeight":0.34,
                                                    "schemaId": "1X9oodXQQKlahX2pYO8P7g.3",
                                                    "diversificationType":"quantitative"
                                  }]
                }
              ] }
         ] }
   }
}
```


Figure 4-12 Example of diversification Configuration

4.6.2 Diversification Manager:

It's the heart of the entire algorithms that produce the diversification score for all the relevance and diversify algorithm.

4.6.2.1 Diversification Base

In this module have the following functionalities:

- Populate all data objects: Populate data, schema, and configuration with semantic rules.
- Calculate Combination and diversity score: it's responsible to create a $(n-1) * n$ combination for all the data n than calculate the all possible diversification score between each tuple using Diversification Score Calculation.
- Semantic Rule: Populate the entire semantic object from the configuration.

4.6.2.2 Diversification Score Calculation:

It's responsible to check the type of the diversification than forward the request to the concern module, i.e.

4.6.2.2.1Categorical:

Let $A_i^{j_{i,1}}, \dots, A_i^{j_{i,d_i}}$ be a subset of d_i semantic rule of relation $\boldsymbol{R_i}$ and $\boldsymbol{v}_i(\tau) = \left[a_i^{j_{i,1}}, \dots, a_i^{j_{i,d^i}}\right]^T$ projection of a combination τ on such rule. Categorical diversity is defined as follows:

$$\delta(\tau_u, \tau_v) = 1 - \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\, \boldsymbol{v}_i(\tau_u) * \omega_i = \boldsymbol{v}_i(\tau_v)$$

Where $\boldsymbol{n}$ is number of categorical relation, $\boldsymbol{\omega_i}$ is the semantic weight, and $\mathbb{1}$ is the indicator function returning one when the predicate is satisfied.

4.6.2.2.2Quantitative:

Let $\mathbf{v}(\tau)$ as defined in categorical diversification. Let $\mathbf{v}(\tau) = [\mathbf{v}_1(\tau), \dots, \mathbf{v}_n(\tau)]^T = [\boldsymbol{v}_1(\tau), \dots, \boldsymbol{v}_d(\tau)]^T$ denote the concatenation of length $d = d_1, \dots, d_n$ of such vectors. Quantitative diversity is defined as follows:

$$\delta(\tau_u, \tau_v) = \sum_{l=1}^{d} \frac{|v_l(\tau_u) - v_l(\tau_v)| - \sigma_{l_{min}}}{\sigma_{l_{max}} - \sigma_{l_{min}}} * \omega_l$$

Where $\sigma_{l_{max}}$ is the maximum difference value of $|v_l(\tau_u) - v_l(\tau_v)|$, $\sigma_{l_{min}}$ is a minimum difference value of $|v_l(\tau_u) - v_l(\tau_v)|$, and $\boldsymbol{\omega_l}$ is the semantic weight of the related rule.

4.6.3 Relevance and Diversification Algorithm

Here we will calculate the top $\mathcal{K}$ tuple based on the relevance and diversification algorithm. Let $N = |\mathcal{R}|$ denote the number of combinations in the result set and $\mathcal{R}_\mathcal{K} \subseteq \mathcal{R}$ the subset of combinations that are presented to the user, where $K = |\mathcal{R}_\mathcal{K}|$. We are interested in identifying a subset $\mathcal{R}_\mathcal{K}$ which is both relevant and diverse. Fixing the relevance score $\mathcal{S}(.,q)$, the dissimilarity function $\delta(.,.)$, and a given integer $\mathcal{K}$, we aim at selecting a set $\mathcal{R}_\mathcal{K} \subseteq \mathcal{R}$ of combinations: Following are the list of algorithm

4.6.3.1 MaxSum

Input : Set of combinations $\mathcal{R}, \mathcal{K}$
Output: Selected combinations $\mathcal{R}_\mathcal{K}$

Begin:
$Define\ \acute{\delta}(\tau_u, \tau_v) = \mathcal{S}(\tau_u, q) + \mathcal{S}(\tau_v, q) + 2\lambda\delta(\tau_u, \tau_v)$

$Initialize\ the\ set\ \mathcal{R}_\mathcal{K} = \phi, U = \mathcal{R}$
$for\ c = 1: \lfloor \mathcal{K}/2 \rfloor$ do
$Find\ (\tau_u, \tau_v) = \underset{x,y \in U}{\operatorname{argmax}}\ \acute{\delta}(x, y)$
$\mathcal{R}_\mathcal{K} = \mathcal{R}_\mathcal{K} \cup \{\tau_u, \tau_v\}$ Set $U = U \setminus \{\tau_u, \tau_v\}$
end

if $\mathcal{K}$ is odd, add an arbitrary combination to $\mathcal{R}_\mathcal{K}$
End:

4.6.3.2 MaxMin

Input : Set of combinations $\mathcal{R}, \mathcal{K}$
Output: Selected combinations $\mathcal{R}_{\mathcal{K}}$

Begin:

$Define\ \acute{\delta}(\tau_u, \tau_v) = \frac{1}{2}\big(\mathcal{S}(\tau_u, q) + \mathcal{S}(\tau_v, q)\big) + \lambda\delta(\tau_u, \tau_v)$

$Initialize\ the\ set\ \mathcal{R}_{\mathcal{K}} = \phi$

$Find\ (\tau_u, \tau_v) = \underset{x,y \in \mathcal{R}}{\text{argmax}}\, \acute{\delta}(x, y)\ and\ set\ \mathcal{R}_{\mathcal{K}} = \{\tau_u, \tau_v\}$

$while\ |\mathcal{R}_{\mathcal{K}}| < \mathcal{K}$ do
    $\tau^* = \underset{\tau \in \mathcal{R} \setminus \mathcal{R}_{\mathcal{K}}}{\text{argmax}}\ min_{x \in \mathcal{R}_{\mathcal{K}}} \acute{\delta}(\tau, x)$
    Set $\mathcal{R}_{\mathcal{K}} = \mathcal{R}_{\mathcal{K}} \cup \{\tau^*\}$
end

End:

4.6.3.3 MMR

Input : Set of combinations $\mathcal{R}, \mathcal{K}$
Output: Selected combinations $\mathcal{R}_{\mathcal{K}}$

Begin:
$Define\ \acute{\delta}(\tau_u, \tau_v) = \big(\mathcal{S}(\tau_u, q) + \mathcal{S}(\tau_v, q)\big)$
$Initialize\ the\ set\ \mathcal{R}_{\mathcal{K}} = \phi$

$Find\ (\tau_u, \tau_v) = \underset{x,y \in \mathcal{R}}{\text{argmax}}\, \acute{\delta}(x, y)\ and\ set\ \mathcal{R}_{\mathcal{K}} = \{\tau_u, \tau_v\}$

$while\ |\mathcal{R}_{\mathcal{K}}| < \mathcal{K}$ do
$\tau^* = \underset{\tau \in \mathcal{R} \setminus \mathcal{R}_{\mathcal{K}}}{\text{argmax}}\ \{\mathcal{S}(\tau, q) + \lambda\, min_{x \in \mathcal{R}_{\mathcal{K}}} \delta(\tau, x)\}$
Set $\mathcal{R}_{\mathcal{K}} = \mathcal{R}_{\mathcal{K}} \cup \{\tau^*\}$
end

End:

Note that for λ = 0 all algorithms return a result set which consists of the top-K combinations with the highest score, thus neglecting diversity.

4.7 Class Diagram

4.7.1 Manipulation Function

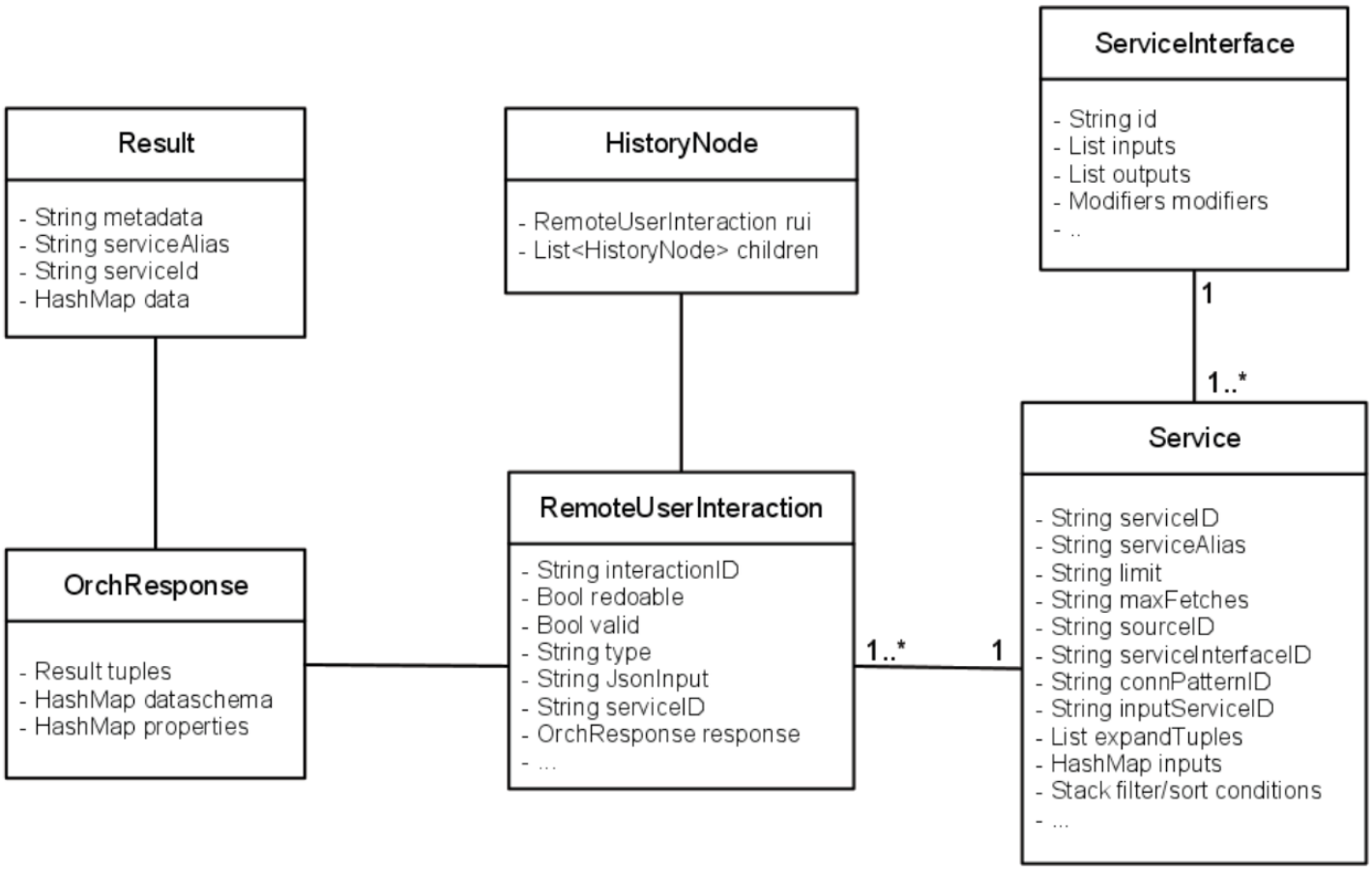


Figure 4-13 Manipulation Function

- **OrchRespose**: it's contained the actual data and contains the objects that contain the following attributes.
    - **Dataschema**: for each service instance (identified by the schema Id) used in the response the list of the outputs (id, type) it produces;
    - **Properties:** hashmap (key/value) of all the response properties
    - **Result:**
        - metadata: json string that contains information about the engine status after invocation (like EOF, invocation time etc)
        - serviceAlias and serviceId of the last service added

    - data: hashmap (id, json string) of all the response tuple

- **History node:** tree that represents the history of the interaction performed by the user. The history contains also invalid nodes (nodes undo-ed) and nodes that even if are invalid can be redo-ed with a redo operation. Type of interactions: FirstQuery, Expand, Filter, Sort, Rerank, Manipulate. Every interaction is performed on a service; a service represents the invocation of a service interface. FirstQuery and Expand create a new service, Filter, Sort, Rerank and Manipulate are performed on old services;

- **List of services:** list of all the services used in the history

- **Map of service interfaces:** map (id, object) of all the service interfaces used in the history.

4.7.2 Diversification score calculation

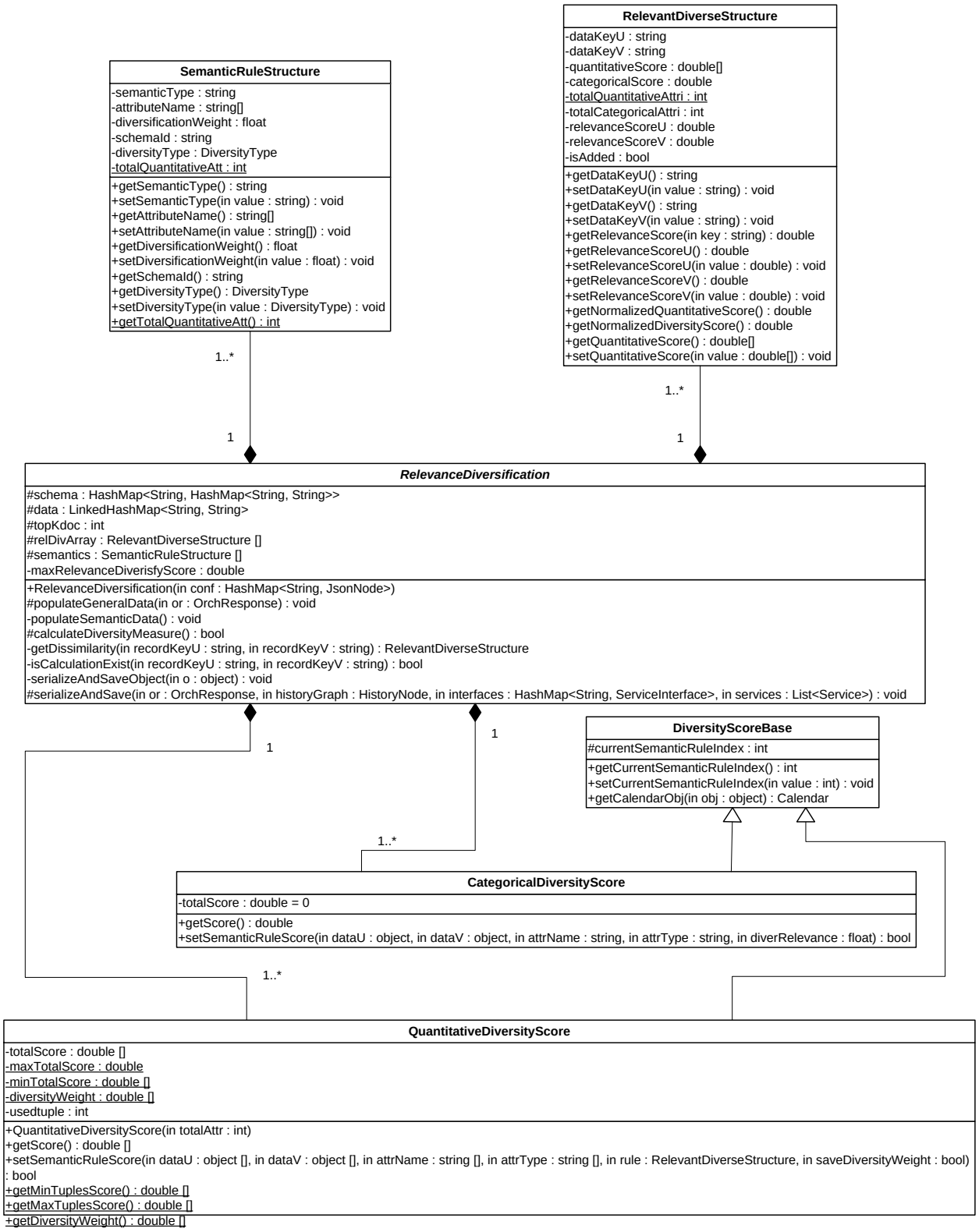


Figure 4-14 Diversification Score Calculation

4.7.2.1 SemanticRuleStructure

Semantic Rules structure mapped to the JSON of Configuration SemanticRules. The attribute of this class has the following meanings.

- **semanticType**: Semantic type of the semantic rule that can either 'SimpleDiff', 'CoordLong', or 'EuclDis'
- **attributeName**: list of all attribute name that a semantic rule needed.
- **diversificationWeight:** Weighting of a semantic rule, range from [0,1], the total weight of all semantic rules should be one.
- **schemaId:** Schema Id where we can see the list of attribute name, in order to calculate the semantic score.
- **diversityType:** Diversification Type of rule, possible value must be 'quantitative' or 'categorical'
- **totalQuantitativeAtt:** Is a static attribute that hold the total number of quantitative semantic rules

This class also hold list of getter/setter method of the all attributes listed above.

4.7.2.2 RelevantDiverseStructure

The core data structure that holds the Semantic Rules structure mapped to the Configuration SemanticRules JSON. The attribute of this class has the following meanings.

- **dataKeyU:** any key from the tuple Id but must be different that dataKeyV.
- **dataKeyV:** any key from the tuple Id but must be different that dataKeyU.
- **quantitativeScore[]:** List of all un normalized score in a Combination of dataKeyU and dataKeyV with respect to the quantitative semantic rules.

- **categoricalScore:** Overall categorical score between dataKeyU and dataKeyV

- **totalQuantitativeAttri:** Static shared attribute, that holds the number of quantitative semantic rule in a diversification manipulation function.

- **totalCategoricalAttri:** Static shared attribute, that holds the number of categorical attribute in a diversification manipulation function.

- **relevanceScoreU:** Relevance score of the dataKeyU

- **relevanceScoreV:** Relevance score of the dataKeyV

- **relevanceDiversifyScore:** Combination of Relevance and diversification score, used for caching the value for the diversification relevance algorithm(e.g. MaxMin).

- **isAdded:** Used for a mark that this combination is added in the result set of the diversification relevance algorithm(e.g. MaxSum)

List of all function other than the getter/setter function is given below:

- **getNormalizedQuantitativeScore:** return the normalized quantitative score by using the global related quantitative minimum and maximum score.

4.7.2.3 RelevanceDiversification

The class is heart for the entire diversification algorithm, sometime called the manager. Responsible for calculating the diversification score that is the backbone of the MaxSum, MaxMin, and MMR algorithms. List of the attribute that use this class is defined below:

- **orchResponse:** Main Object of orchestrator
- **schema:** Schema of the data attribute as define in the orchResponse.
- **data:** Data receives from OrchResponse.data during the invocation of this Manipulation function.
- **topKdoc:** Top k data that is required in the output
- **relDivArray:** All possible combination for Diversification of the available tuple
- **semantics:** Array of all possible semantic rule that is define in the configuration
- **maxRelevanceDiverisfyScore:** If the value is null than its means that maxRelevanceDiverisfyScore is not initialized
- **sortedKeyIndex:** Its hold the next index for in the sort data function.

List of all function other than the getter/setter function is given below:

- **RelevanceDiversification:** Constructor with the diversification manipulation function configuration that is mentioned in the architecture.
- **populateGeneralData:** Initialized all the attribute related to the data and schema.

- **populateSemanticData:** Initialized all the semantic data that contains the rule type, semantic meanings of the rule, list of attribute that participate into the semantic rule, service name where we can find out the services.

- **calculateDiversityMeasure:** main function of this class, responsible for making all the possible combination of the data and calculate the diversification score with the help of CategoricalDiversityScore, and QuantitativeDiversityScore classes

- **getDissimilarity:** get the relevance diversification structure that contain the parameter key U and V, order does not matter**.**

- **isCalculationExist:** return true if the parameter key U and V find in the combination list **relDivArray.**

- **serializeAndSave:** Serialize the input request and save it on the physical location in order to do the application debugging.

- **setData:** Add the additional attribute in the result set of the MaxMin, MaxSum, MMR algorithms.

- **sortData:** Sort the result set based on the diverisicaiton relevance score, helping method for MaxMin, MaxSum, MMR algorithms.

4.7.2.4 DiversityScoreBase

DiversityScoreBase is an abstract class that helps CategoricalDiversityScore and QuantitativeDiversityScore to do their processing. List of the attribute that use this class is defined below:

- **currentSemanticRuleIndex:** current semantic rule index based on their type (i.e. Categorical, Quantitative).

List of all function other than the getter/setter function is given below:

- **getCalendarObj:** it's a helper function that convert the string data into calendar object in order to do the attribute type based processing.

4.7.2.5 CategoricalDiversityScore

CategoricalDiversityScore is a specialized class of the DiversityScoreBase that calculates diversification score for the semantic rule that has the diversification type equals to categorical. List of the attribute that use this class is defined below:

- **totalScore:** Un-Normalized categorical diversification score related to a data combination for the semantic rule.

List of all function other than the getter/setter function is given below:

- **getScore:** gives us a normalized categorical diversification score related to a data combination.

- **setSemanticRuleScore:** Update the total score if the data of the semantic rule match with each other.

4.7.2.6 QuantitativeDiversityScore

QuantitativeDiversityScore is a specialized class of the DiversityScoreBase that calculates quantitative diversification score for the semantic rule that has the diversification type equals to categorical. List of the attribute that use this class is defined below:

- **totalScore:** Array of quantitative tuple score related to a data combination for each semantic rule with respect to their definition order in the configuration.

- **maxTotalScore:** Static List of Maximum quantitative semantic rule score. If there is any quantitative semantic rule in the call of this manipulation function, than after setting each and every quantitative score of the data combination we will find here the maximum score of the quantitative semantic rule with respect to their definition order in the configuration.

- **minTotalScore:** Static List of Minimum quantitative attribute score. If there is any quantitative semantic rule in the call of this manipulation function, than after setting each and every quantitative score of the data combination we will find here the

minimum score of the quantitative semantic rule with respect to their definition order in the configuration.

- **diversityWeight:** Static List of quantitative semantic rule Weight. If there is any quantitative semantic rule in the call of this manipulation function, than after setting each and every quantitative score of the data combination we will find here the quantitative semantic rule Weight of the quantitative semantic rule with respect to their definition order in the configuration.

- **usedtuple:** Current quantitative semantic rule index number. In other word, it's the order of the processed quantitative semantic rule that is define in the configuration

List of all function other than the getter/setter function is given below:

- **setSemanticRuleScore:** Set the quantitative semantic rule score for a given data combination.

4.7.3 Computing Relevant and Diverse Combinations

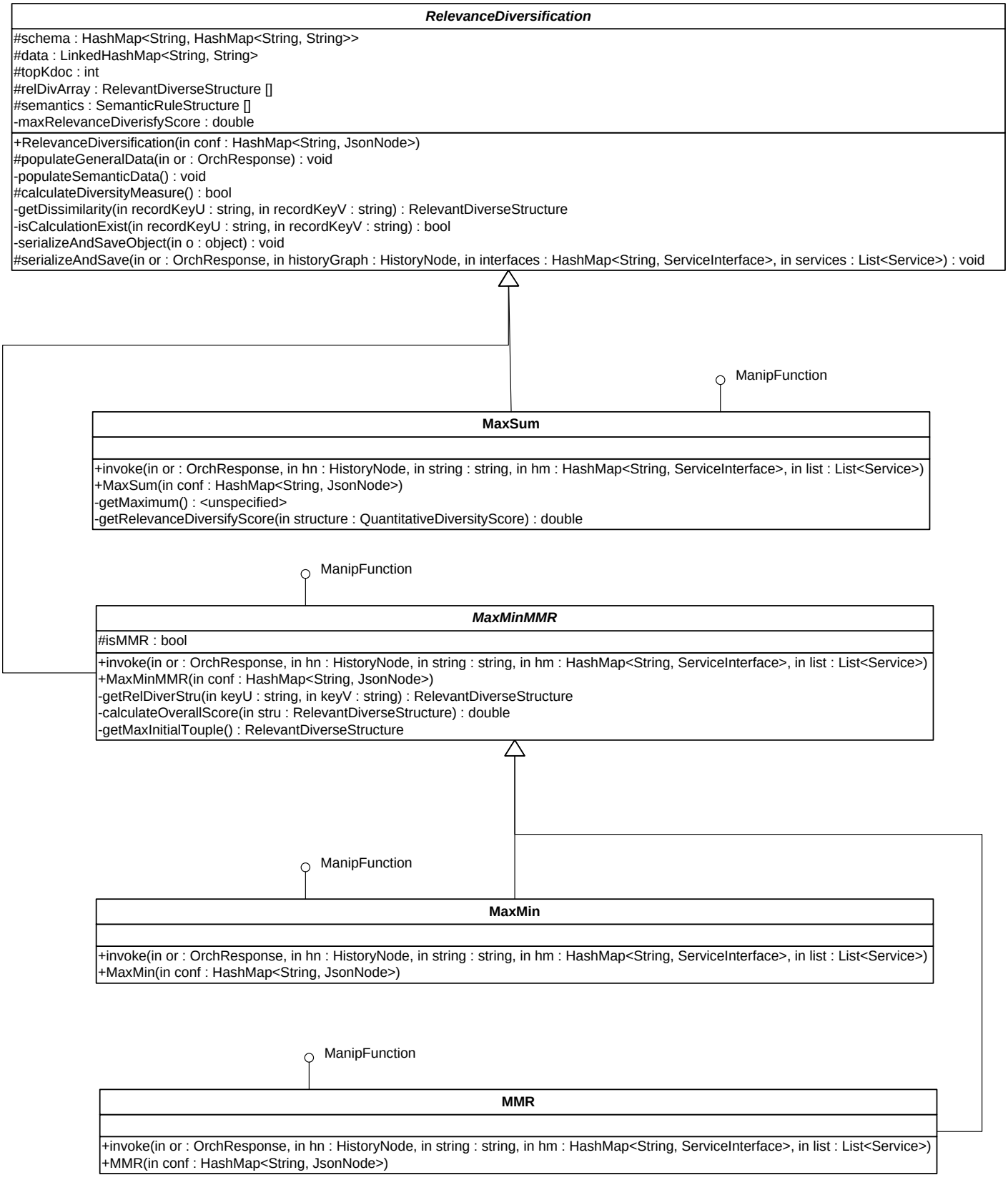


Figure 4-15 Computing Relevant and Diverse Combinations

4.7.3.1 MaxSum

MaxSum updates the ranking of the tuple based on the relevance and diversify score of the data set using MaxSum Pseudo code. List of the attribute that use this class is defined below:

- **docIndex:** It's an index number that hold the ranking of the tuple during processing for the result set.

List of all function other than the getter/setter function is given below:

- **getMaximum**: Get the data combination that has not yet been added in the result set based on the maximum score (calculated by getRelevanceDiversifyScore function). After getting, add the combination into the result set and mark all the combination that has either data key U or V.

- **getRelevanceDiversifyScore:** Calculate a data combination score based on their relevance and diversification score

- **invoke:** It's an implementation of the manipulation function interface using MaxSum pseudo code in order provide update the tuple score ranking and return the top K tuple.

4.7.3.2 MaxMinMMR

MaxMinMMR is an abstract class that updates the ranking of the tuple based on the relevance and diversifies score of the data set using MaxMin and MMR Pseudo code. List of the attribute that use this class is defined below:

- **isMMR:** It's a flag that indentify which algorithm need to use during the processing.

List of all function other than the getter/setter function is given below:

- **getRelDiverStru:** Return the Relevance Diversification structure based on parameter Key U and V.

- **getMaxInitialtuple:** It select the first combination that has the max score based on the isMMR flag.

- **calculateOverallScore:** Calculate the score of the RelevantDiverseStructure if isMMR flag is equal to false.

4.7.3.3 MaxMin

It's a specialized class of MaxMinMMR, and implements the interface of the Manipulation Function. In the constructor, it tells to its generalized class to do the processing based on MaxMin functionality.

4.7.3.4 MMR

It's a specialized class of MaxMinMMR, and implements the interface of the Manipulation Function. In the constructor, it tells to its generalized class to do the processing based on MMR functionality.

Chapter 5. Implementation

5.1 Building RelevantDiverseStructure Array

Building RelevantDiverseStructure array is the backbone of any relevance diversification algorithm (e.g. MaxSum, MaxMin, and MMR). The major responsibility of this pseudo code is to calculate the diversification between each combination of the data tuple

5.1.1 Input:

The following objects are the list of the input that require to invoke this functionality

1. Set of Semantic rule S of type **SemanticRuleStructure,** which is populated from the manipulation function configuration file.
2. Set of data tuple, that is the part of the tuple attribute of the OrchResponse class

5.1.2 Output

The following objects are the list of the output that we will get from this functionality:

1. Array of **RelevantDiverseStructure** object**:** which represent the all possible combination of the data tuple
2. Array of MaximumQuantitativeScore: if there is any quantitative semantic rule in the call of this manipulation function, than you will find here the maximum score of the quantitative semantic rule with respect to their definition order in the configuration.
3. Array of MinimumQuantitativeScore: if there is any quantitative semantic rule in the call of this manipulation function, than you will find here the minimum score of the quantitative semantic rule with respect to their definition order.
4. Array of WeightQuantitative: if there is any quantitative semantic rule in the call of this manipulation function, than you will find here the rule weight of the quantitative semantic rule with respect to their definition order in the configuration.

How the function computes require functionality is described on the next page:

```
MaximumQuantitativeScore[] = new Double (SemanticRuleStructure.getTotalQuantitativeAtt())
MinimumQuantitativeScore[] = new Double (SemanticRuleStructure.getTotalQuantitativeAtt())
WeightQuantitative [] = new Double (SemanticRuleStructure.getTotalQuantitativeAtt())
IsIntialized = FALSE
 totalTouples = this.data.size()
totalCombination = ((totalTouples - 1) * (totalTouples)) / 2
relDivArray = new RelevantDiverseStructure( totalCombination)
FOR (Integer indexI = 0, indexI < data.length; indexI++) THEN
        FOR (Integer indexJ = indexI +1, indexJ < data.length; indexJ++) THEN
                quantitativeIndex = 0,  categoricalScore = 0
                quantitativeScore = new Double (SemanticRuleStructure.getTotalQuantitativeAtt())
                FOR EACH semantic in S
                        FOR EACH attribute in semantic->AtributeNameList
                                Get the attribute with value form the corresponding schema
                        END FOR
                        Calculate the score based on the semanticType
                        IF sematic.diversificationType EQUAL 'categorical'
                                IF attribute value are same THEN
                                        categoricalScore = categoricalScore +
semantic.diversificationWeight
                                End IF
                        ELSE
                                currentScore = Calculate the Semantic Quantitative Score
                                IF IsIntialized == FALSE THEN
                                        MaximumQuantitativeScore [quantitativeIndex] = currentScore
                                        MinimumQuantitativeScore [quantitativeIndex] = currentScore
                                        WeightQuantitative [quantitativeIndex] = semantic. diversiWeight
                                ELSE
                                        IF
MaximumQuantitativeScore[quantitativeIndex]<currentScoreTHEN
                                        MaximumQuantitativeScore [quantitativeIndex] = currentScore
                                ENDIF
                                IF  MinimumQuantitativeScore [quantitativeIndex] > currentScore THEN
                                        MinimumQuantitativeScore [quantitativeIndex] = currentScore
                                ENDIF
                        END IF
                        quantitativeScore [quantitativeIndex]  = currentScore
                        quantitativeIndex = quantitativeIndex + 1
                END IF
                END FOR
                relDivArray[usedId++] = new RelevantDiverseStructure(data[indexI].key, data[indexJ].key
         , data[indexI].RelevanceScore, data[indexJ].RelevanceScore,  quantitativeScore, categoricalScore);
         END FOR
END FOR
```

Figure 5-1 Building RelevantDiverseStructure Array

5.2 Get Normalized Diversification Score

This functionality helps to find out the normalized diversification score.

5.2.1 Input

The following objects is the list of the input that require to invoke this functionality

1. An Object of **RelevantDiverseStructure** for which you want to find out the diversification score.
2. Array of MaximumQuantitativeScore: if there is any quantitative semantic rule in the call of this manipulation function, than we have to give the maximum score array for quantitative semantic rules with respect to their definition order in the configuration.
3. Array of MinimumQuantitativeScore: if there is any quantitative semantic rule in the call of this manipulation function, than we have to give the minimum score array for quantitative semantic rules with respect to their definition order in the configuration.
4. Array of WeightQuantitative: if there is any quantitative semantic rule in the call of this manipulation function, than we have to give the rule weight array for quantitative semantic rules with respect to their definition order in the configuration.

5.2.2 Output

The following object is the output that we will get from this functionality:

1. Value of the normalized diversification score for give object

How the function computes require functionality is described given below:

```
score = R. categoricalScore
FOR ( index = 0; index <R.quantitativeScore.length; index++ )
        score + = (( (R.quantitativeScore[index] - MIN[index])   / ( MAX[index] - MIN[index]) )  * W[index]);
END FOR
```

Figure 5-2 Get Normalized Diversification Score

5.3 MaxSum

Update the ranking of the data, and also restrict the size of the data to top **K** number of elements.

5.3.1 Input

The following objects are the list of the input that require to invoke this functionality

1. Set of Semantic rule S of type **SemanticRuleStructure,** which is populated from the manipulation function configuration file.
2. Set of data tuple, that is the part of the tuple attribute of the OrchResponse class
3. Number of the element **K** that we want to see in the result set
4. Diversification coefficient

5.3.2 Output

The following object is the output that we will get from this functionality:

1. Updated data with the relevance and diversify score
2. Only top **K** element in the result set that has the maximum score computed from MaxSum algorithm.

How the function computes require functionality is described on the next page:

```
Build the RelevantDiverseStructure Array /// from the previous algorithm
docIndex = 1
FOR ( index = 0, index < (K/2); index++) THEN
        RelevantDiverseStructure selected = NOTHING
        FOR EACH RelevantDiverseStructure R IN RelevantDiverseStructure Array
                IF NOT R.ISADDED THEN
                        IF (selected IS NOTHING)  THEN
                                Selected =R
                        ELSE
                                currentScore =
(Selected.getRelevanceScoreU()+Selected.getRelevanceScoreV()
                                        + 2 * Lambda * Selected.getNormalizedDiversityScore());
                                newScore = (R.getRelevanceScoreU() +R.getRelevanceScoreV()
                                        + 2 * Lambda * R.getNormalizedDiversityScore());
                                IF (newScore <  previousScore) THEN
                                        Selected =R
                                END IF
                        END IF
                END IF
        END FOR
        FOR EACH RelevantDiverseStructure R IN RelevantDiverseStructure Array
                IF R. getDataKeyU  == selected. getDataKeyU OR R. getDataKeyU  == selected. getDataKeyV
                OR R. getDataKeyV  == selected. getDataKeyU OR R. getDataKeyV  == selected. getDataKeyV

                        SET R.IS ADDED = TRUE
                END IF
        END FOR
        Set the docIndex and docIndex+1 data document index number for selected.getDataKeyU
         & selected.getDataKeyV based on their relevance score.
END FOR
```

Figure 5-3 MaxSum Pseudo Code

5.4 Get RelevantDiverseStructure

This functionality helps MaxMin and MMR functionality to find out RelevantDiveriseStructure

5.4.1 Input

The following objects is the list of the input that require to invoke this functionality

1. dataKeyU: A tuple Id for which we have to find out the combination. It must be different than the dataKeyV.
2. dataKeyV: A tuple Id for which we have to find out the combination. It must be different than the dataKeyV.
3. Array of RelevantDiverseStructure: It's a populated array from Building RelevantDiverseStructure Array functionality, and we have to find out the a RelevantDiverseStructure that have dataKeyU and dataKeyV.

5.4.2 Output

The following object is the output that we will get from this functionality:

1. RelevantDiverseStructure that has the dataKeyU and dataKeyV.

How the function computes require functionality is described given below:

```
FOR ( index = 0; index < relDivArray.length; index++ )
IF ((relDivArray[index].getDataKeyU().equals(dataKeyU)  AND
relDivArray[index].getDataKeyV().equals(dataKeyV))   || (relDivArray[index].getDataKeyU().equals(dataKeyV)
&& relDivArray[index].getDataKeyV().equals(dataKeyU)))
THEN
            return relDivArray[index];
END FOR
```

Figure 5-4 Get RelevantDiverseStructure

5.5 MaxMin

Update the ranking of the data, and also restrict the size of the data to top **K** number of elements.

5.5.1 Input

The following objects are the list of the input that require to invoke this functionality

1. Set of Semantic rule S of type **SemanticRuleStructure,** which is populated from the manipulation function configuration file.
2. Set of data tuple, that is the part of the tuple attribute of the OrchResponse class
3. Number of the element **K** that we want to see in the result set
4. Diversification coefficient

5.5.2 Output

The following object is the output that we will get from this functionality:

1. Updated data with the relevance and diversify score
2. Only top **K** element in the result set that has the maximum score computed from MaxMin algorithm.

How the function computes require functionality is described below:

```
Build the RelevantDiverseStructure Array /// from the previous algorithm
String[] addedDataKey
RelevantDiverseStructure selected = NOTHING

FOR EACH RelevantDiverseStructure R IN RelevantDiverseStructure Array
        IF selected IS NOTHING OR (0.5 * (selected.getRelevanceScoreU() + selected.getRelevanceScoreV())
         + Lambda * selected.getNormalizedDiversityScore()) THEN
                selected = R
        END IF
END FOR
```

```
Set the first and second data document index number for selected.getDataKeyU  & selected.getDataKeyV
based on their relevance score.
addedDataKey [0] = selected. getDataKeyU
addedDataKey [1] = selected. getDataKeyV

WHILE  (docIndex < K)
        candidateKeyTotalScore = 0 ,
        candidateKey = NOTHING
        FOR EACH Key IN data.keySet
                alreadyAdded = False
                FOR  (addedKeyIndex = 0; addedDataKey[addedKeyIndex] != null; addedKeyIndex++)
                        IF key IS EQUAL addedDataKey[addedKeyIndex] THEN
                                alreadyAdded =  TRUE
                                BREAK FOR
                        END IF
                END FOR
                selected =  NOTHING;
                IF NOT alreadyAdded THEN
                        currentScore = -1
                        FOR  (addedKeyIndex = 0; addedDataKey[addedKeyIndex] != null; addedKeyIndex++)
                                tempDataStru = getRelDiverStru(oldKey, addedDataKey[addedKeyIndex]);
                                tempScore = (0.5 * (stru.getRelevanceScoreU + stru.getRelevanceScoreV)
                                        + Lambda * stru.getNormalizedDiversityScore)
                                IF currentScore == -1 OR tempScore < currentScore THEN
                                        currentScore = tempScore
                                        selected  = tempDataStru
                                END IF
                        END FOR
                        IF (currentScore > candidateKeyTotalScore)  THEN
                                candidateKeyTotalScore = currentScore
                                candidateKey = key
                        END IF
                END IF
        END FOR
        addedDataKey[docIndex++] = candidateKey

         Update the data document index number with docIndex where data key is candidateKey
END FOR
```

Figure 5-5 MaxMin Pseudo Code

5.6 MMR

Update the ranking of the data, and also restrict the size of the data to top **K** number of elements.

5.6.1 Input

The following objects are the list of the input that require to invoke this functionality

1. Set of Semantic rule S of type **SemanticRuleStructure,** which is populated from the manipulation function configuration file.
2. Set of data tuple, that is the part of the tuple attribute of the OrchResponse class
3. Number of the element **K** that we want to see in the result set
4. Diversification coefficient

5.6.2 Output

The following object is the output that we will get from this functionality:

1. Updated data with the relevance and diversify score
2. Only top **K** element in the result set that has the maximum score computed from MMR algorithm.

How the function computes require functionality is described below:

```
Build the RelevantDiverseStructure Array /// from the previous algorithm
String[] addedDataKey
RelevantDiverseStructure selected = NOTHING

FOR EACH RelevantDiverseStructure R IN RelevantDiverseStructure Array
        IF selected IS NOTHING OR ((selected.getRelevanceScoreU + selected.getRelevanceScoreV )
         < (R.getRelevanceScoreU + R.getRelevanceScoreV) THEN
                selected = R
        END IF
END FOR
```

```
Set the first and second data document index number for selected.getDataKeyU  & selected.getDataKeyV
based on their relevance score.
addedDataKey [0] = selected. getDataKeyU
addedDataKey [1] = selected. getDataKeyV

WHILE  (docIndex < K)
        candidateKeyTotalScore = 0
        candidateKey = NOTHING
        FOR EACH Key IN data.keySet
                alreadyAdded = False
                FOR  (addedKeyIndex = 0; addedDataKey[addedKeyIndex] != null; addedKeyIndex++)
                        IF key IS EQUAL addedDataKey[addedKeyIndex] THEN
                                alreadyAdded =  TRUE
                                BREAK
                        END IF
                END FOR
                selected =  NOTHING;
                IF NOT alreadyAdded THEN
                        currentScore = -1
                        RelevantDiverseStructure dataStruMMR = NOTHING;
                        FOR  (addedKeyIndex = 0; addedDataKey[addedKeyIndex] != null; addedKeyIndex++)
                                tempDataStru = getRelDiverStru(oldKey, addedDataKey[addedKeyIndex]);
                                tempScore = (0.5 * (stru.getRelevanceScoreU + stru.getRelevanceScoreV)
                                        + Lambda * stru.getNormalizedDiversityScore)
                                IF currentScore == -1 OR tempScore < currentScore THEN
                                        currentScore = tempScore
                                        selected  = tempDataStru
                                END IF
                        END FOR
                        IF (dataStruMMR.getRelevanceScore(key) + Lambda * currentScore)
                        > candidateKeyTotalScore THEN
                                candidateKeyTotalScore = dataStruMMR.getRelevanceScore(key)
                                                        + Lambda * currentScore
                                candidateKey = key
                        END IF
                END IF
        END FOR
        addedDataKey[docIndex++] = candidateKey

         Update the data document index number with docIndex where data key is candidateKey
END FOR
```

Figure 5-6 MMR Pseudo Code

Chapter 6. Evaluation

In the last chapter, we did the implementation of MaxSum, MaxMin, and MMR algorithm. Now, we do the evaluation of these three algorithms in order to check which algorithm is good and which is not. For this purpose, we checked the accuracy based on the data set generated from SeCo application, and complexity for solving the diversity problem.

6.1 Accuracy

Search computing accuracy is based on novelty-relevance, and recall measurement. In this thesis, we did an implementation of $\alpha - DCG_k$ and $\mathrm{MD} - Recall_k$ (formally $\mathrm{S} - \mathrm{Recall}_k$) in java that is fully integrated with the SeCo data type to check novelty-relevance and recall measurement for the diversification algorithm.

6.1.1 α-Discounted Cumulative Gain

A well-known metrics for relevance in diversified result sets is the α-Discounted Cumulative Gain ($\alpha - DCG_k$) [19], which measures the usefulness (gain) of a document based on its position in the result list and its novelty w.r.t. the previous results in the ranking. In the original formulation of $\alpha - DCG_k$, documents are composed by a set of information nuggets. In the context of multi-domain result-sets, the $\alpha - DCG_k$ can be defined by assimilating an information nugget to a tuple $t_i \in R_i$, where $R_1, \dots, R_n$ are the relations involved in a multi-domain query. Therefore, we define $\alpha - DCG_k$ as:

$$\alpha - \mathrm{DCG_K} = \sum_{\mathrm{k=1}}^{\mathrm{K}} \frac{\sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{J}(\tau_\mathrm{k}, \mathrm{t_i})(1-\alpha)^{\mathrm{r_{t_i},k-1}}}{\log_2(1+\mathrm{k})}$$

Where $J(\tau_k, t_i)$ returns 1 when tuple t_i appears in a combination τ_k position k in the result set and o otherwise. J is defined as:

$$\mathrm{J}(\tau_\mathrm{k}, \mathrm{t_i}) = \mathbb{I}\pi_{\mathrm{R_i}}(\tau_\mathrm{k}) = \mathrm{t_i}$$

where π_{R_i} denotes the projection over the attributes ofR_i, and $r_{t_i,k-1} = \sum_{j=1}^{k-1} J(\tau_j, t_i)$ quantifies the number of combinations up to position $k-1$ that contain the tuple t_i.

6.1.2 MD- Recall

Sub-topic recall at rank K ($S - Recall_k$) [20] is a recall measure for search results related to several sub-topics, often applied to evaluate diversification algorithms [14]. In multi domain search, a tuple in a combination can be assimilated to a subtopic in a document. Therefore, multi-domain recall at rank K ($MD - Recall_k$), defined next, measures, for each relation R_i involved in a query (with $i = 1 \dots n$) and for all rank positions k from 1 to K, the set of distinct tuples $\left(R_i^k = \{t_i \in R_i | \exists j \leq k . \pi R_i(\tau_j) = t_i\}\right)$ retrieved in the result set, with respect to the entire population of the relation ($|R_i|$)

$$MD - Recall_K = \prod_{i=1}^{n} \frac{\left|\bigcup_{k=1}^{n} R_i^k\right|}{|R_i|}$$

We have implemented the MD-Recall functionality in the SeCo architecture and the pseudo code is the following.

Input: OrchResponse or, double alpha
Output: List of the score value

```
FOR ( index = 0; index < relDivArray.length; index++ )
IF ((relDivArray[index].getDataKeyU().equals(dataKeyU)  AND
relDivArray[index].getDataKeyV().equals(dataKeyV))   || (relDivArray[index].getDataKeyU().equals(dataKeyV)
&& relDivArray[index].getDataKeyV().equals(dataKeyU)))
THEN
            return relDivArray[index];
END FOR
```

Figure 6-1 MD-Recall

6.1.3 Dataset

We generated a dataset from SeCo by limit the number of results for each request to five. In other word, we set "**limit = 5**" in the orchestrator properties that can be find in the following file.
\orchestrator\WEB-INF\conf\ orch.properties

After this, we select each and every Hotel and Restaurant during the expansion.

Dataset consisting of three relations: Hotel (5 tuples), Restaurant (5 tuples), and Museum (5 tuples), that is shown below:

Hotel

Name	Category	Average rating	Lowest Price	Tuple Score
Hotel Center 1-2-3	Hotel	6.5	62.0	1.0
Hotel Marsala	Hotel	7.4	75.0	0.99000
Hotel Stromboli	Hotel	7.8	50.0	0.98000
B & B La Basilica	B&B	7.8	40.0	0.97000
Hotel Torino	Hotel	8.1	90.0	0.95999

Table 6-1 Hotel Data Set

Restaurant

Name	Category	Average Rating	Average Price	Tuple Score
La paella 2	Spain	2.0	75.0	1.0
Il giardino degli aranci	Italian	2.8	50.0	0.99000
Aroma di Pechino	Cinese	3.0	15.0	0.98000
Centrale Ristotheatre	Italian	3.0	20.0	0.97000
Doozo	Japanese	3.5	20.0	0.95999

Table 6-2 Restaurant Data Set

Museum

Name	Category	Full fee	Reduced Fee	Tuple Score
Galleria Borghese	Artistic	6.5	3.25	1.0
Galleria Doria Pamphilj	Artistic	7.3	5.7	0.99000
Galleria Nazionale d'Arte Moderna	Artistic	6.2	3.1	0.98000
Galleria Nazionale d'Arte Antica	Artistic	6.0	3.5	0.97000
Galleria Spada	Artistic	5.0	2.5	0.95999

Table 6-3 Museum Data Set

The 125 combinations have been produced by making a Cartesian production of five Hotel, Restaurant and Museum.

The global relevance score for the hotel, museum and restaurant is calculated based on the average, mathematically

$$= \left(\frac{Hotel.TupleScore + Resaurant.TupleScore + Museum.TupleScore}{3}\right)$$

6.1.4 Categorical Diversification

Suppose, we want only categorical diversification based on hotel name, restaurant name, and museum name. Then our semantic rule can be like the following:

```
        {"semanticType": "SimpleDiff",
                "attributeName":["name"],
                "diversificationWeight":0.33,
                "schemaId": "9Mip3B1IpOnFHJPZNJD3CH.1",
                "diversificationType":"categorical"
        },
        {"semanticType": "SimpleDiff",
                "attributeName":["name"],
                "diversificationWeight":0.33,
                "schemaId": "GjvxAw2xhlnTPEx33A6SeK.2",
                "diversificationType":"categorical"
        },
        {"semanticType": "SimpleDiff",
                "attributeName":["name"],
                "diversificationWeight":0.34,
                "schemaId": "1X9oodXQQKlahX2pYO8P7g.3",
                "diversificationType":"categorical"
        }
]
```

Figure 6-2 Example Categorical Diversification JSON

$\alpha - DCG_k$ and $MD - Recall_k$ for the result sets obtained with no diversification and with the diversification algorithms MMR, MaxSum, and MaxMin, applying categorical distances. Each data point of the X-axis represents the k-th element in the result-set. The Y-axes represent, respectively, the values of $\alpha - DCG_k$ and $MD - Recall_k$

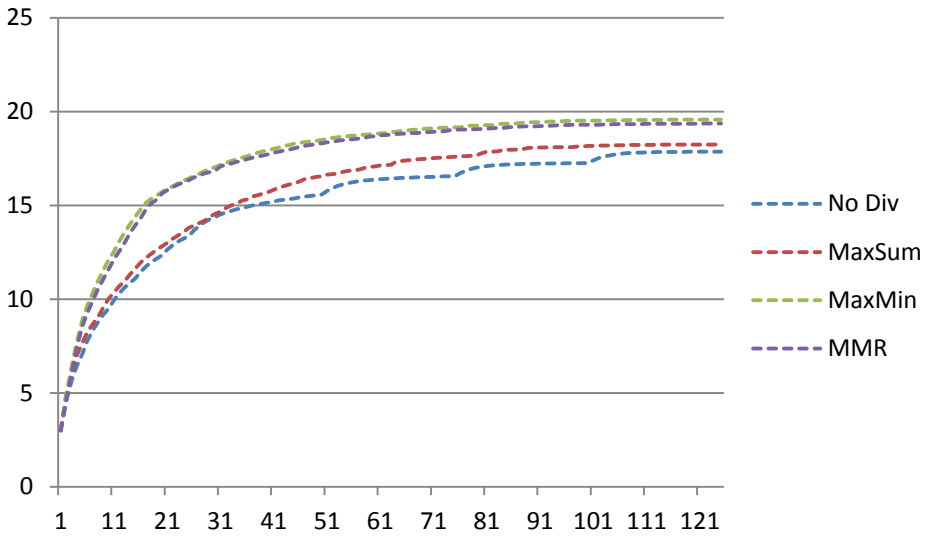


Figure 6-3 α-DCG Categorical Diversification (α = 0.25, λ = 0.25)

When algorithm's diversification coefficient has a lower value and $\alpha - DCG_k$ novelty element has a low value compare to relevance than MMR, MaxMin perform a little bit good from the base line while MaxSum has almost the same result as the base line.

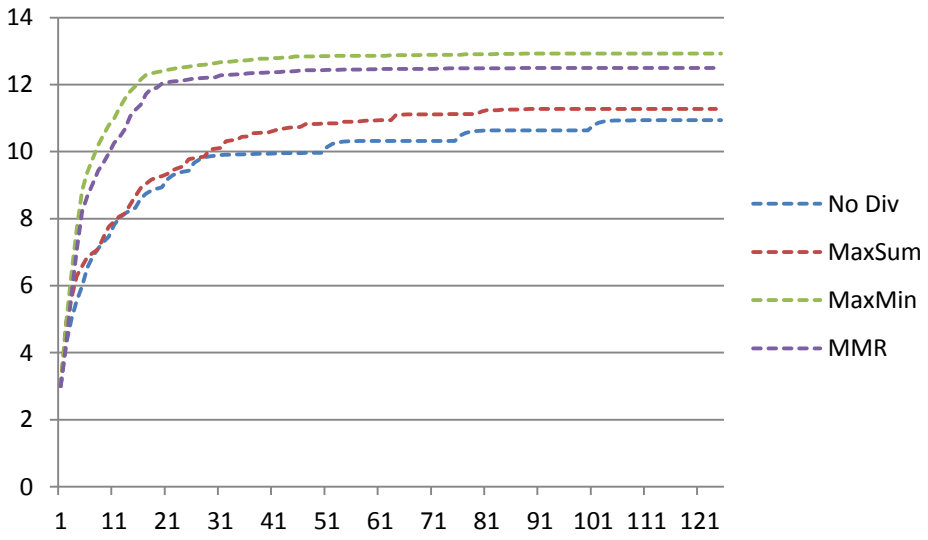


Figure 6-4 α-DCG Categorical Diversification (α = 0.5, λ = 0.25)

When algorithm's diversification coefficient has a lower value and $\alpha - DCG_k$ has equally novelty and relevance than MMR, MaxMin gives a very good result from the base line while MaxSum has almost the same result as the base line.

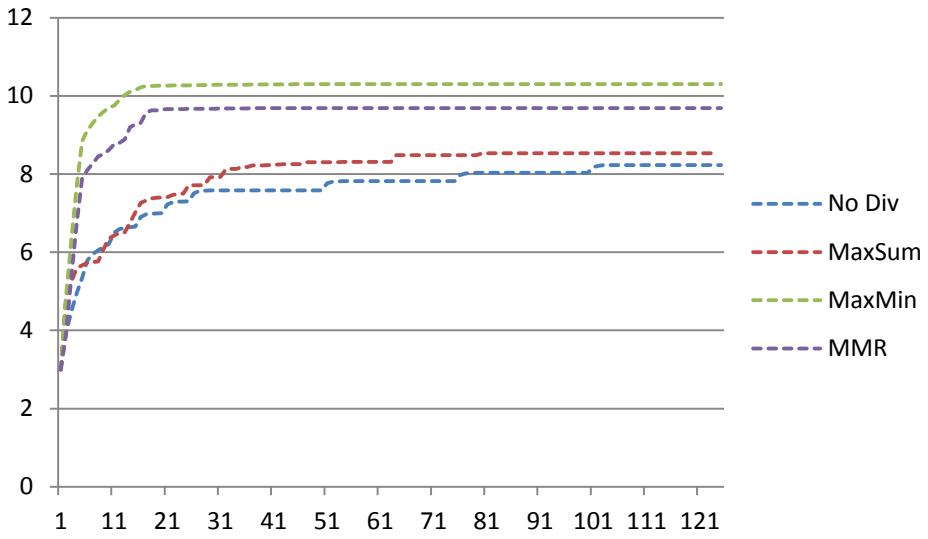


Figure 6-5 α-DCG Categorical Diversification (α = 0.75, λ = 0.25)

When algorithm's diversification coefficient has a lower value and $\alpha - DCG_k$ has greater novelty compare to relevance than MaxMin perform a very good results and MMR perform less than MaxMin but good from the base line while MaxSum has a very little performance compare to the base line.

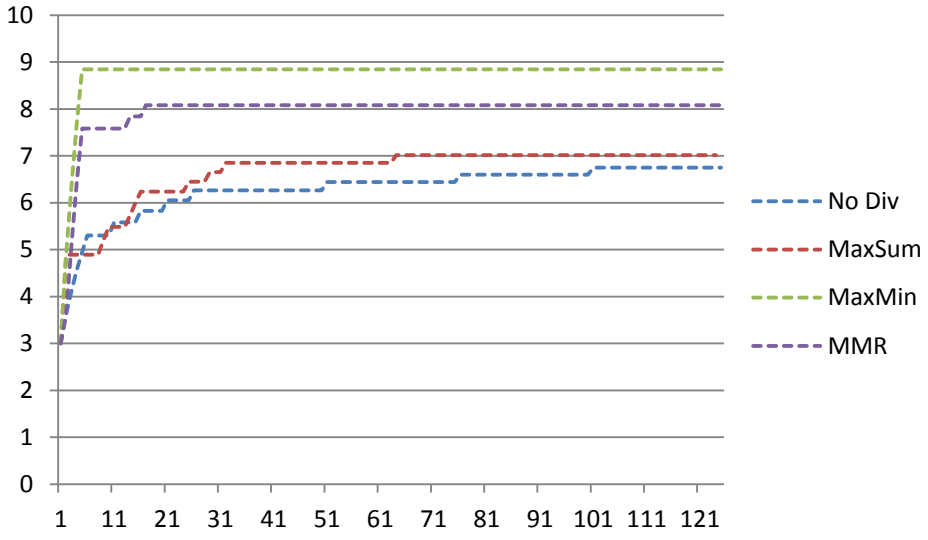


Figure 6-6 α-DCG Categorical Diversification (α = 1.0, λ = 0.25)

When algorithm's diversification coefficient has a lower value and $\alpha - DCG_k$ has greatest novelty compare to relevance than MaxMin perform an excellent results and MMR perform less than MaxMin but very good from the base line while MaxSum has a very little performance compare to the base line.

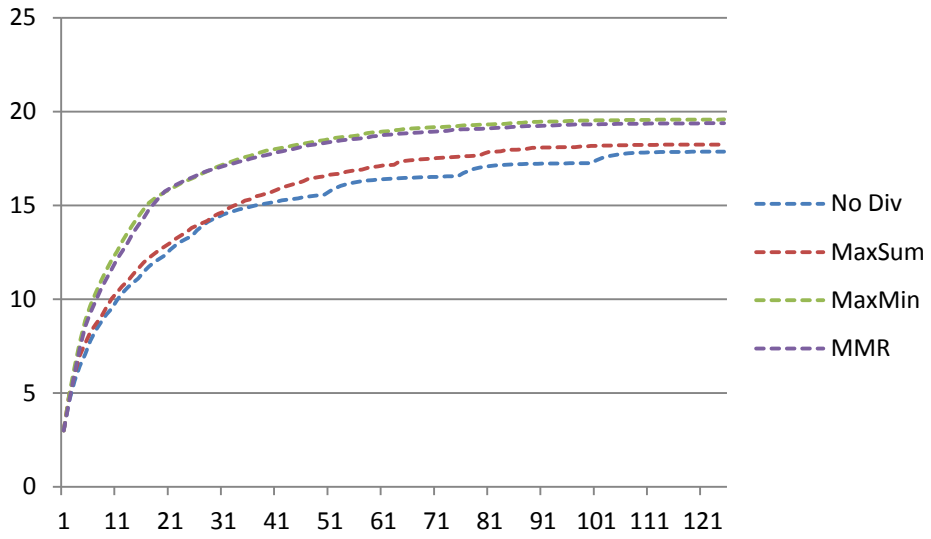


Figure 6-7 α-DCG Categorical Diversification (α = 0.25, λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance and $\alpha - DCG_k$ *novelty element has a low* value compare to relevance than MMR and MaxMin perform a little bit good from the base line while MaxSum has almost the same result as the base line.

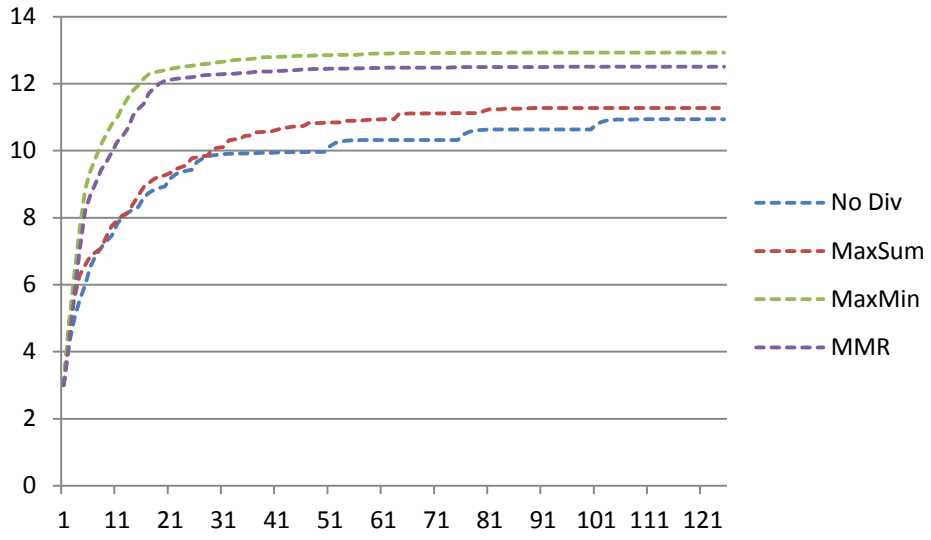


Figure 6-8 α-DCG Categorical Diversification (α = 0.5, λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance and $\alpha - DCG_k$ has equally novelty and relevance than MMR, MaxMin gives a very good from the base line while MaxSum has a very little performance compare to the base line.

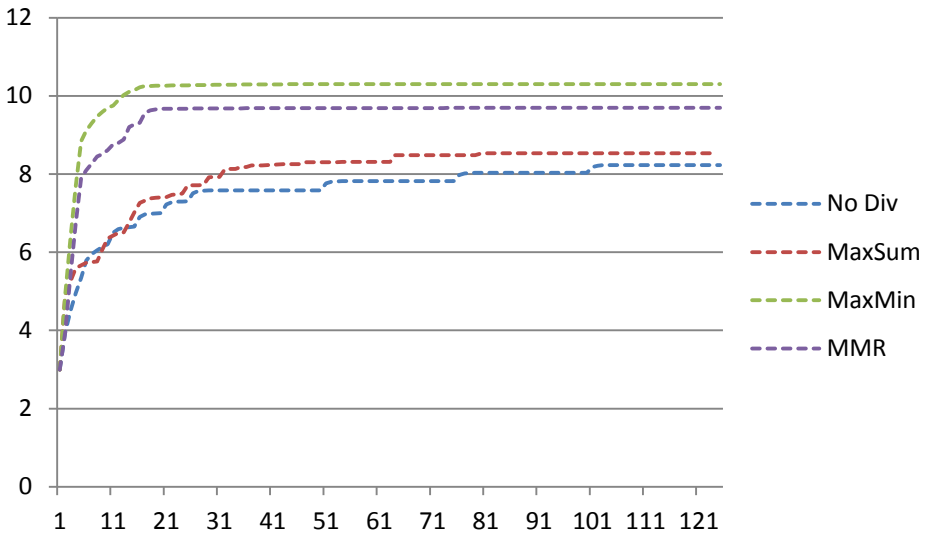


Figure 6-9 α-DCG Categorical Diversification (α = 0.75, λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance and $\alpha - DCG_k$ has greater novelty compare to relevance then MaxMin perform very good results and MMR perform little less than MaxMin but good from the base line while MaxSum give a very little performance compare to the base line.

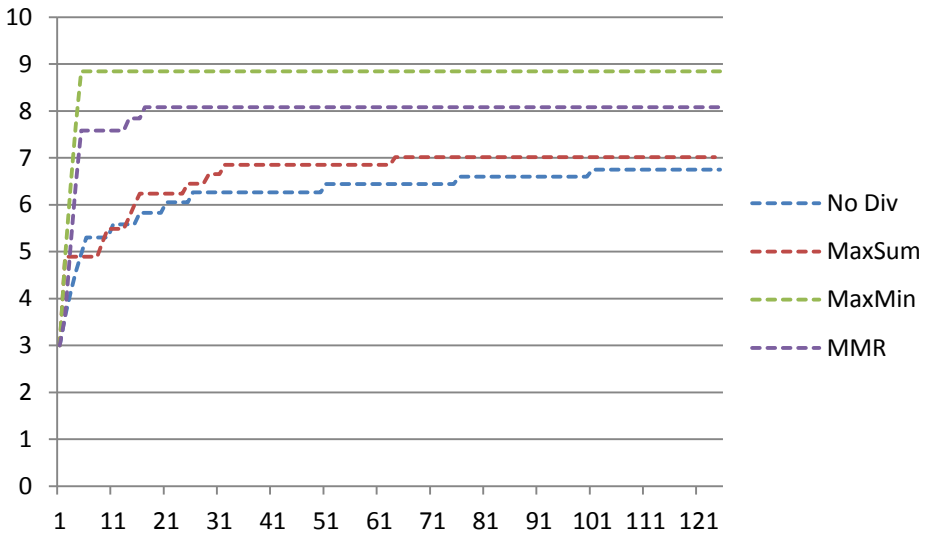


Figure 6-10 α-DCG Categorical Diversification (α = 1.0, λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance and $\alpha - DCG_k$ has greatest novelty compare to relevance than MaxMin perform an excellent results and MMR perform very less than MaxMin but very good from the base line while MaxSum has a very little performance compare to the base line.

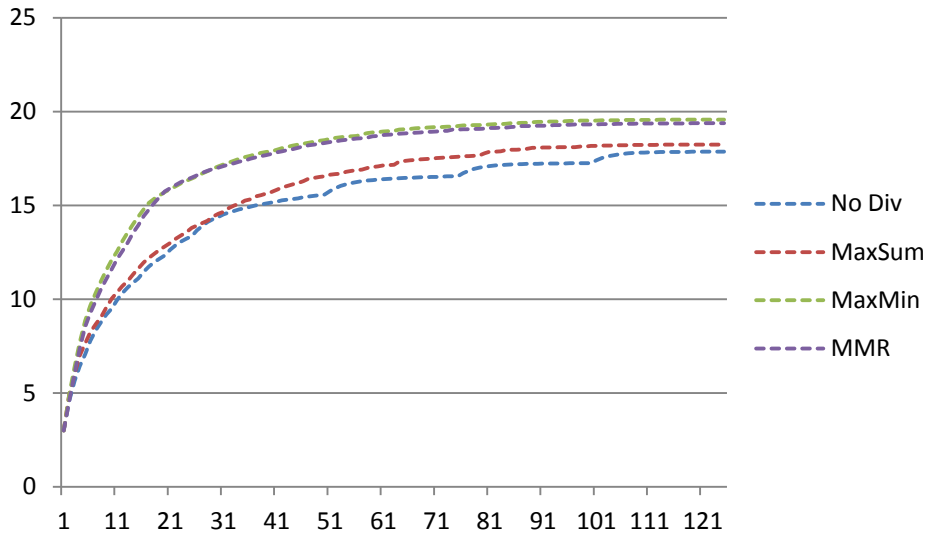


Figure 6-11 α-DCG Categorical Diversification (α = 0.25, λ = 0.75)

When algorithm's diversification coefficient has greater value as the relevance and $\alpha - DCG_k$ *novelty element has a low* value compare to relevance than MMR and MaxMin perform a little bit good from the base line while MaxSum has almost the same result as the base line.

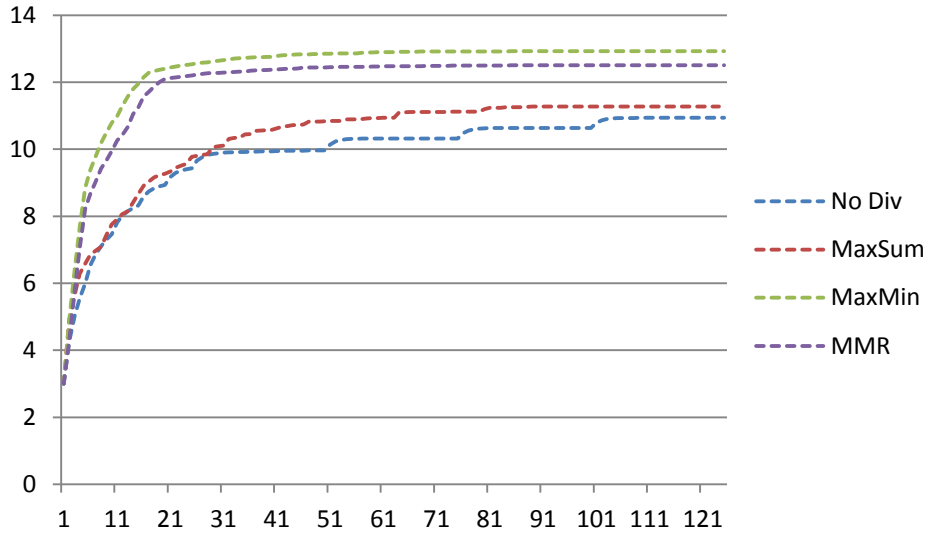


Figure 6-12 α-DCG Categorical Diversification (α = 0.5, λ = 0.75)

When algorithm's diversification coefficient has greater value as the relevance and $\alpha - DCG_k$ has equally novelty and relevance than MMR, MaxMin gives a very good from the base line while MaxSum has a very little performance compare to the base line.

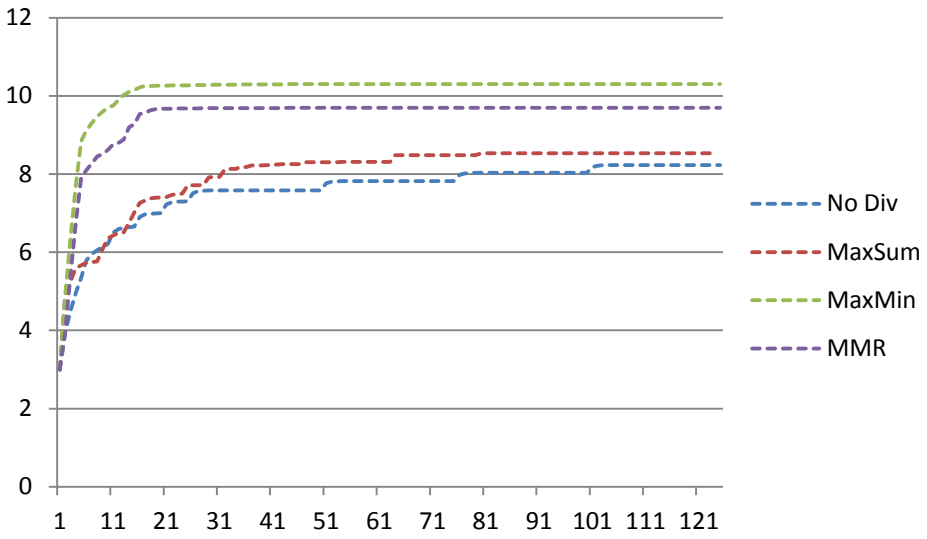


Figure 6-13 α-DCG Categorical Diversification (α = 0.75, λ = 0.75)

When algorithm's diversification coefficient has greater value as the relevance and $\alpha - DCG_k$ has greater novelty compare to relevance than MMR perform a very good results and MaxMin perform little less than MMR but good from the base line while MaxSum has a very little performance compare to the base line.

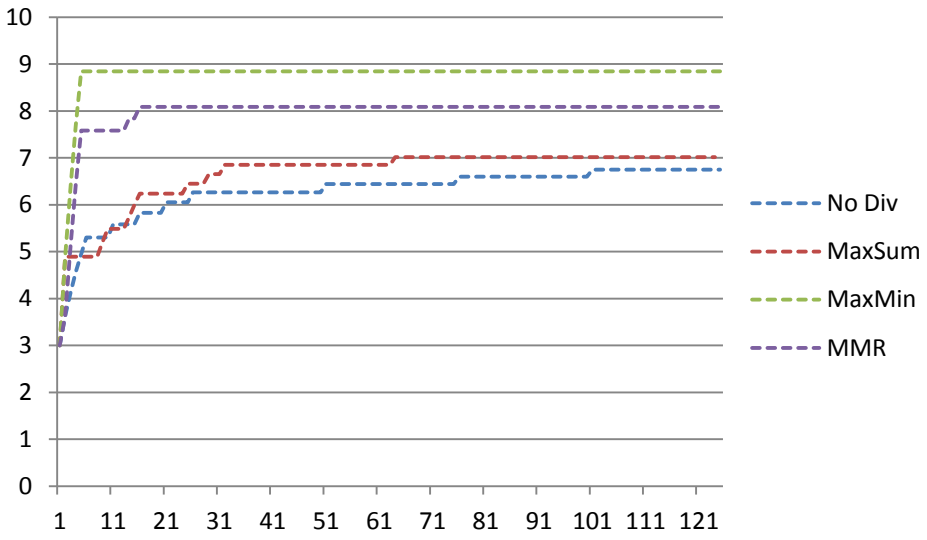


Figure 6-14 α-DCG Categorical Diversification (α = 1.0, λ = 0.75)

When algorithm's diversification coefficient has greater value as the relevance and $\alpha - DCG_k$ has greatest novelty compare to relevance than MMR perform an excellent results and MaxMin perform less than MMR but very good from the base line while MaxSum has a very little performance compare to the base line.

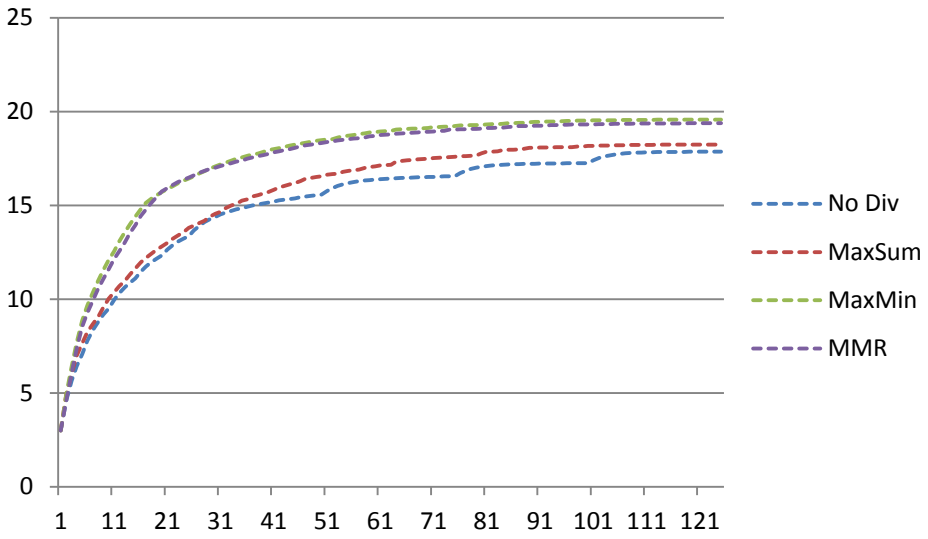


Figure 6-15 α-DCG Categorical Diversification (α = 0.25, λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance and $\alpha - DCG_k$ *novelty element has a low* value compare to relevance than MMR and MaxMin perform a little bit good from the base line while MaxSum has almost the same result as the base line.

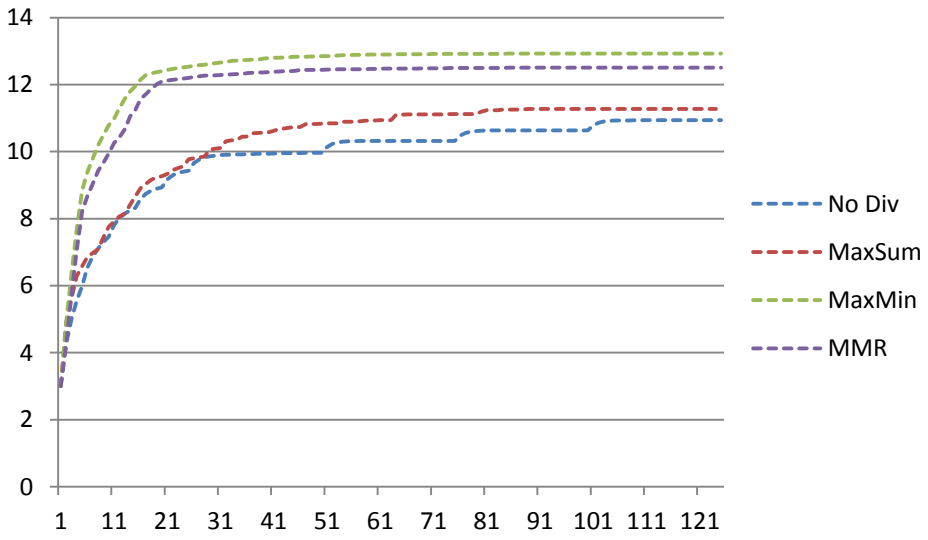


Figure 6-16 α-DCG Categorical Diversification (α = 0.5, λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance and $\alpha - DCG_k$ has equally novelty and relevance than MaxMin gives a very good result and MMR perform good result from the base line while MaxSum had almost the same result as the base line.

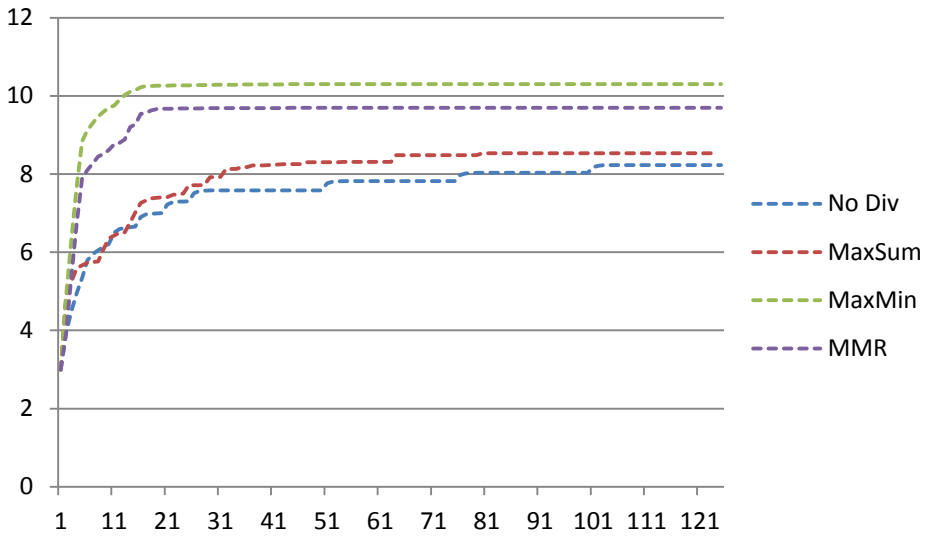


Figure 6-17 α-DCG Categorical Diversification (α = 0.75, λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance and $\alpha - DCG_k$ has greater novelty compare to relevance than MMR perform a very good results and MaxMin perform little less than MMR but good from the base line while MaxSum has a very little performance compare to the base line.

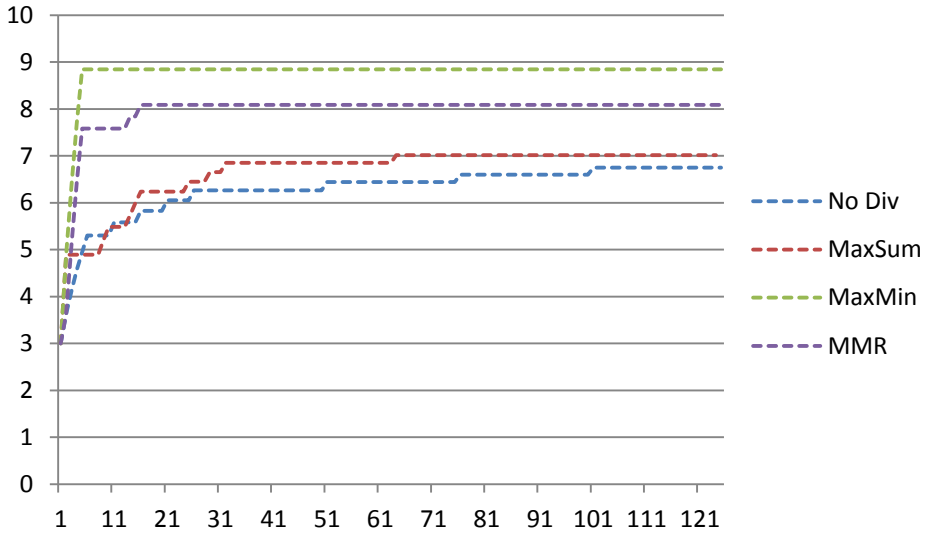


Figure 6-18 α-DCG Categorical Diversification (α = 1.0, λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance and $\alpha - DCG_k$ has greatest novelty compare to relevance than MaxMin perform an excellent results and MMR perform less than MaxMin but very good from the base line while MaxSum has a very little performance compare to the base line.

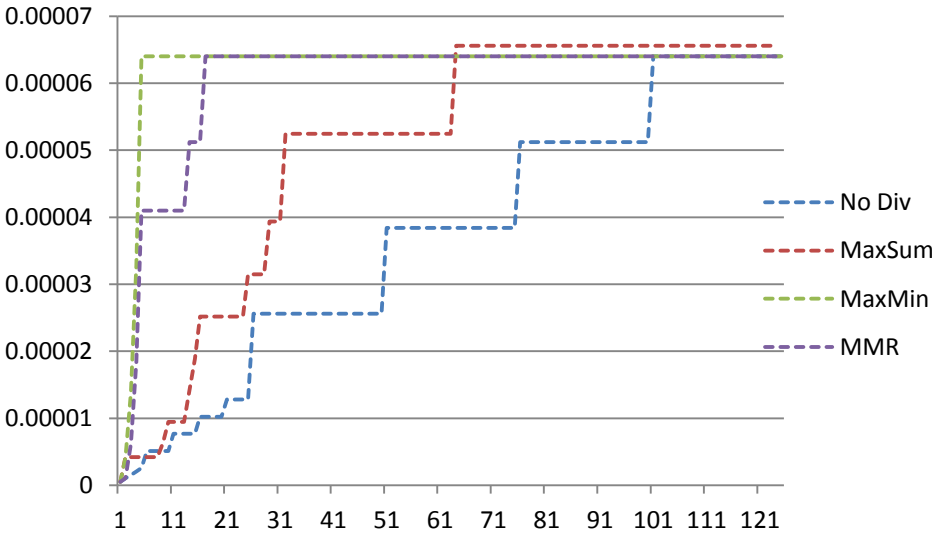


Figure 6-19 MD-Recall Categorical Diversification (λ = 0.25)

When algorithm's diversification coefficient has a lower value than MMR and MaxMin gives good performance from the base line while MaxSum offers only small performance compare to MMR and MaxMin.

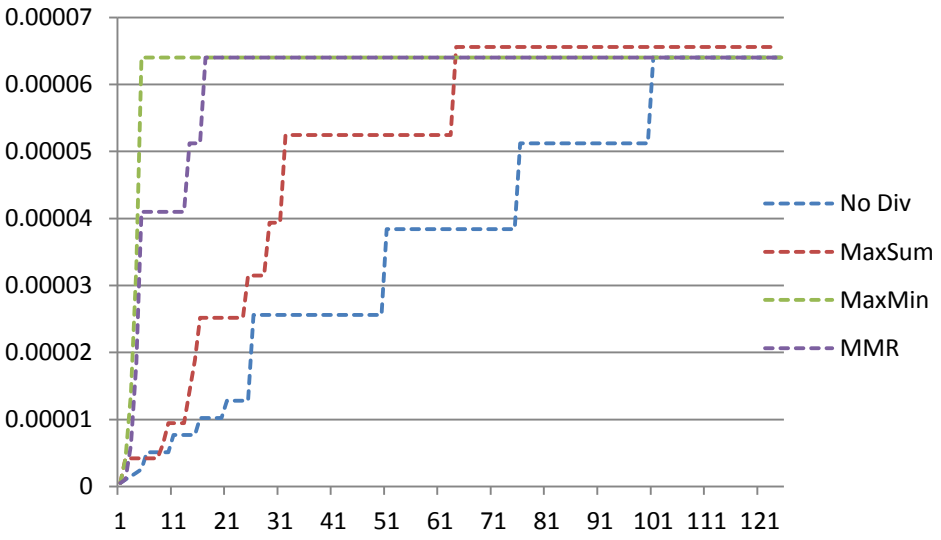


Figure 6-20 MD-Recall Categorical Diversification (λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance than MMR and MaxMin gives good performance from the base line while MaxSum offers only small performance compare to MMR and MaxMin.

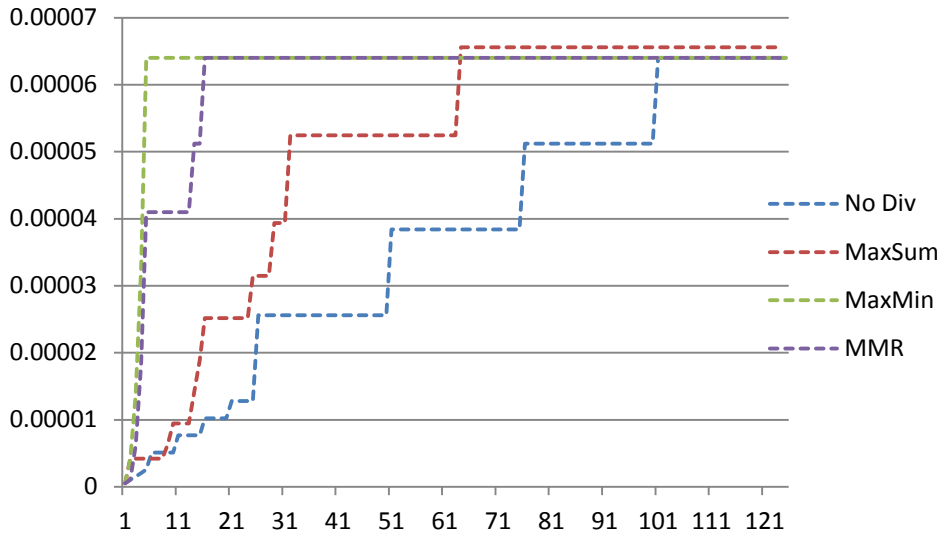


Figure 6-21 MD-Recall Categorical Diversification (λ = 0.75)

When algorithm's diversification coefficient has greater value as the relevance than MMR and MaxMin gives good performance from the base line while MaxSum offers only small performance compare to MMR and MaxMin.

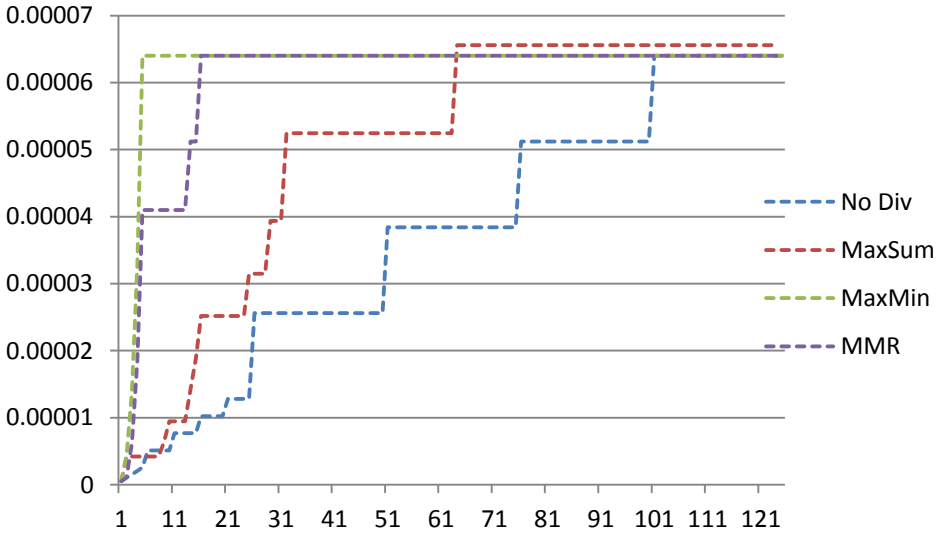


Figure 6-22 MD-Recall Categorical Diversification (λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance than MMR and MaxMin gives good performance from the base line while MaxSum offers only small performance compare to MMR and MaxMin.

6.1.5 Quantitative Diversification

Suppose we want only quantitative diversification on hotel lowest price, restaurant average price, and museum full fee. Then our semantic rule can be like this:

```
[ {"semanticType": "SimpleDiff",
                    "attributeName":["lowestPrice"],
                    "diversificationWeight":0.33,
                    "schemaId": "9Mip3B1IpOnFHJPZNJD3CH.1",
                    "diversificationType":"quantitative"
          },
          {"semanticType": "SimpleDiff",
                    "attributeName":["avgPrice"],
                    "diversificationWeight":0.33,
                    "schemaId": "GjvxAw2xhlnTPEx33A6SeK.2",
                    "diversificationType":"quantitative"
          },
          {"semanticType": "SimpleDiff",
                    "attributeName":["fullFee"],
                    "diversificationWeight":0.34,
                    "schemaId": "1X9oodXQQKlahX2pYO8P7g.3",
                    "diversificationType":"quantitative"
          }
]
```

Figure 6-23 Example Quantitative Diversification JSON

$\alpha - DCG_k$ and $MD - Recall_k$ for the result sets obtained with no diversification and with the diversification algorithms MMR, MaxSum, and MaxMin, applying quantitative distances. Each data point of the X-axis represents the k-th element in the result-set. The Y-axes represent, respectively, the values of $\alpha - DCG_k$ and $MD - Recall_k$

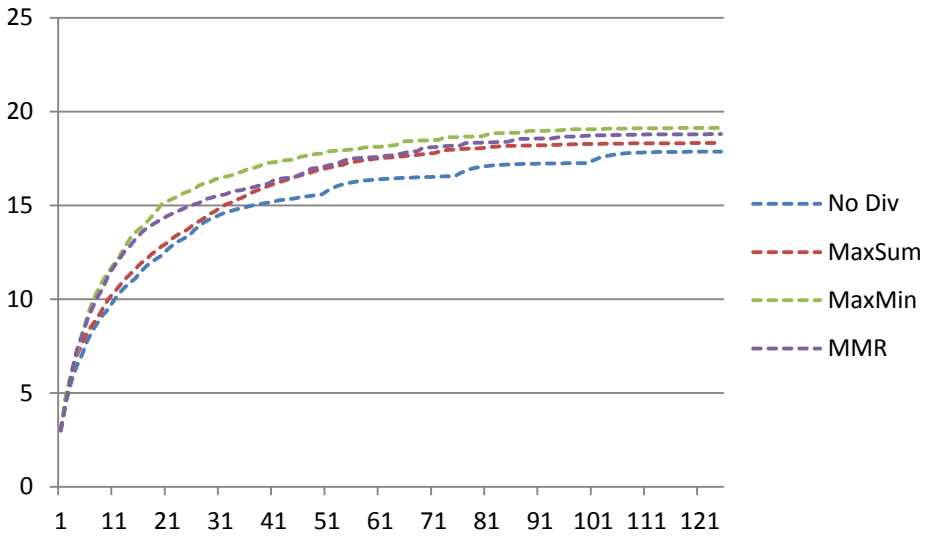


Figure 6-24 α-DCG Quantitative Diversification (α = 0.25, λ = 0.25)

When algorithm's diversification coefficient has a lower value and $\alpha - DCG_k$ *novelty element has a low* value compare to relevance than MMR MaxMin, and MaxSum perform a little bit good from the base line but MMR and MaxMin perform better than MaxSum.

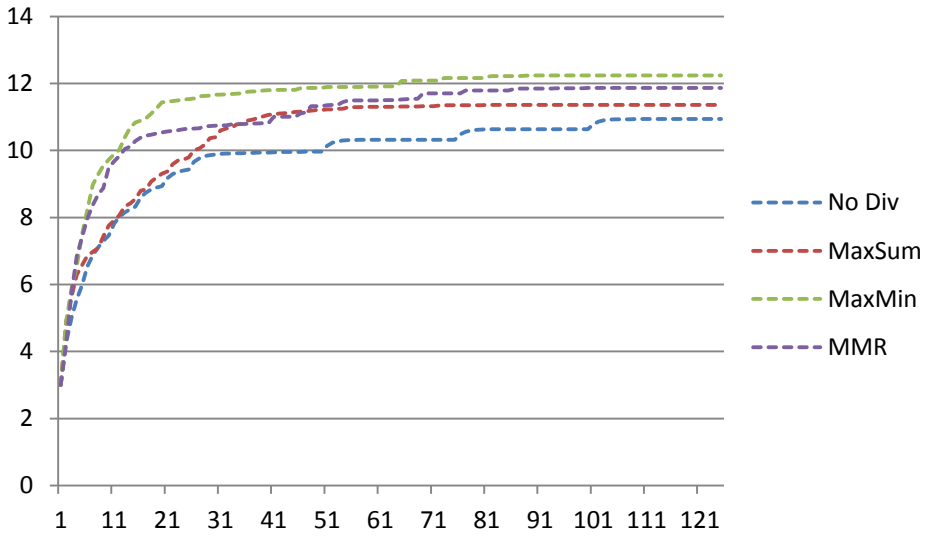


Figure 6-25 α-DCG Quantitative Diversification (α = 0.5, λ = 0.25)

When algorithm's diversification coefficient has a lower value and $\alpha - DCG_k$ has equally novelty and relevance than MaxMin, MMR gives a good result from the base line while MaxSum perform better than the base and almost has equal to MMR.

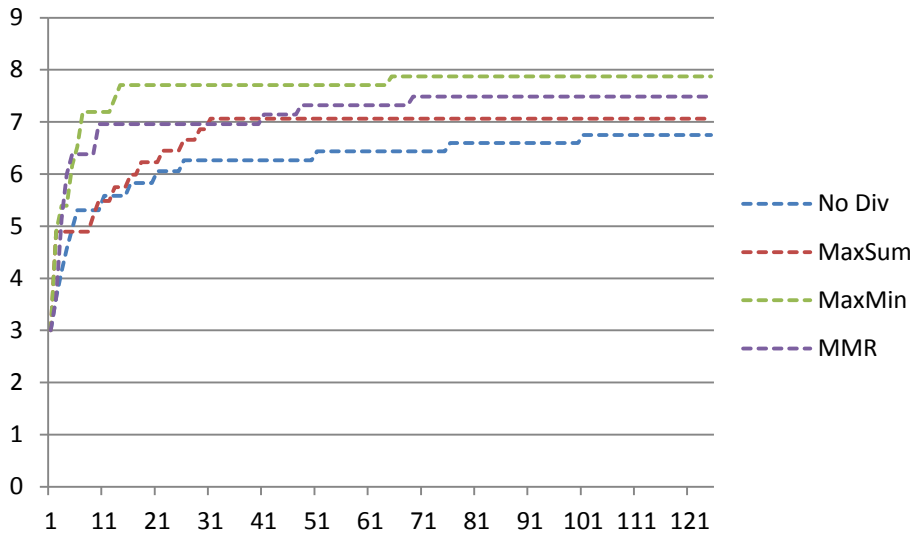


Figure 6-26 α-DCG Quantitative Diversification (α = 0.75, λ = 0.25)

When algorithm's diversification coefficient has a lower value and $\alpha - DCG_k$ has greater novelty compare to relevance than MMR perform a very good results and MaxMin perform less than MMR but good from the base line while MaxSum has a little performance compare to the base line and MMR, MaxMin.

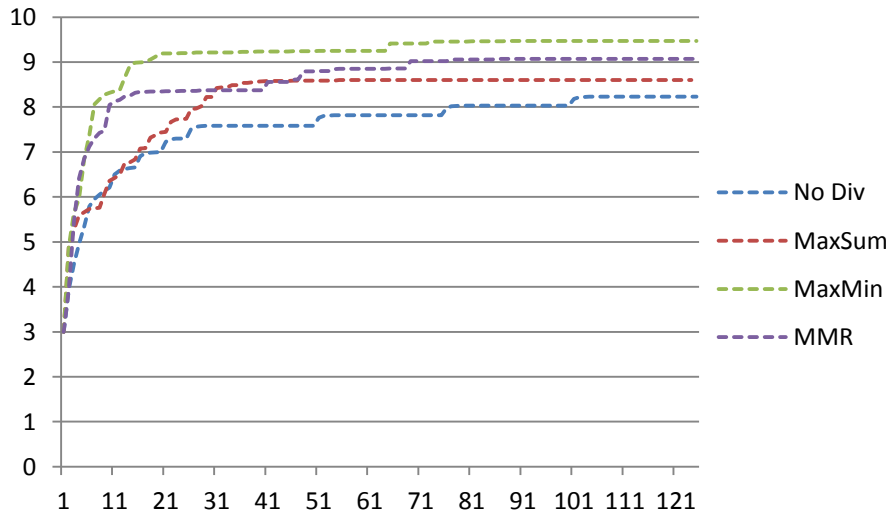


Figure 6-27 α-DCG Quantitative Diversification (α = 1.0, λ = 0.25)

When algorithm's diversification coefficient has a lower value and $\alpha - DCG_k$ has greatest novelty compare to relevance than MMR perform an excellent results and MaxMin perform very less than MMR but very good from the base line while MaxSum has a very little performance compare to the base line.

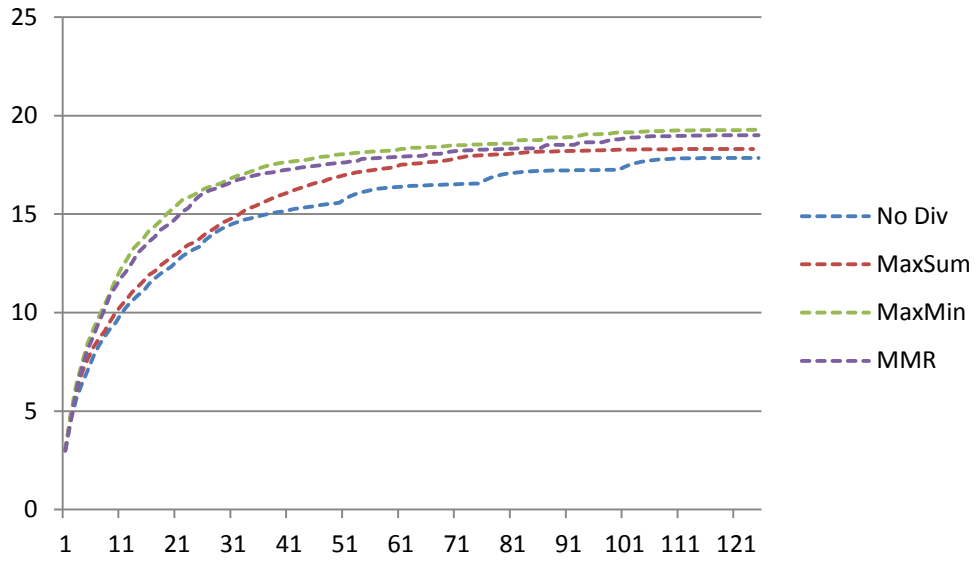


Figure 6-28 α-DCG Quantitative Diversification (α = 0.25, λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance and $\alpha - DCG_k$ novelty element has a low value compare to relevance than MMR MaxMin, and MaxSum perform a little bit good from the base line but MMR and MaxMin perform better than MaxSum.

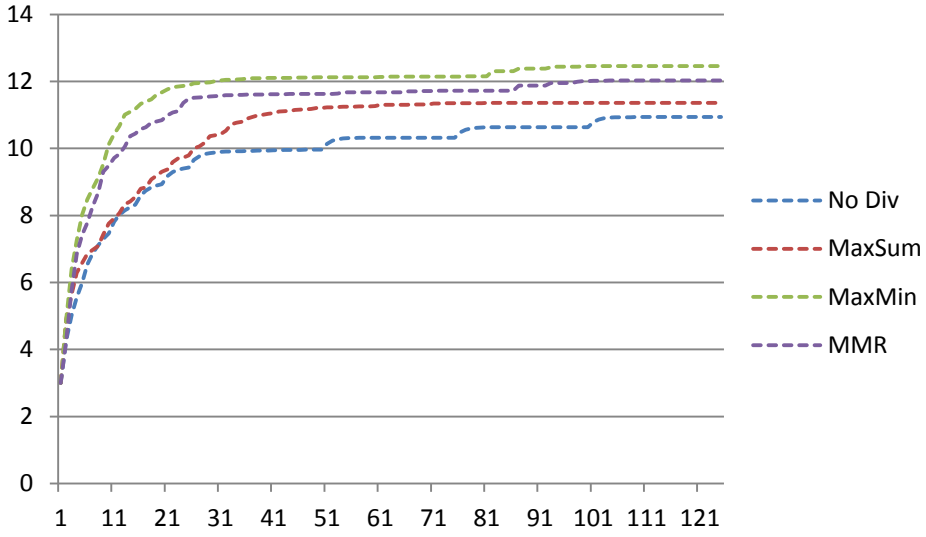


Figure 6-29 α-DCG Quantitative Diversification (α = 0.5, λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance and $\alpha - DCG_k$ has equally novelty and relevance than MaxMin, MMR gives a good result from the base line while MaxSum has also a good performance than the base line but less than MaxSum and MMR.

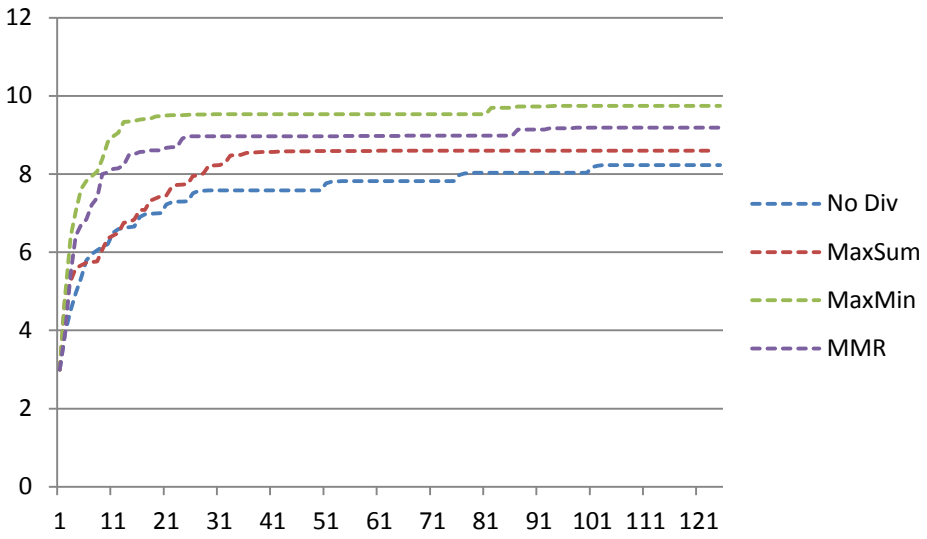


Figure 6-30 α-DCG Quantitative Diversification (α = 0.75, λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance and $\alpha - DCG_k$ has greater novelty compare to relevance than MMR perform a very good results and MaxMin perform less than MMR but good from the base line while MaxSum has a performance compare to the base line but less than MMR, MaxMin.

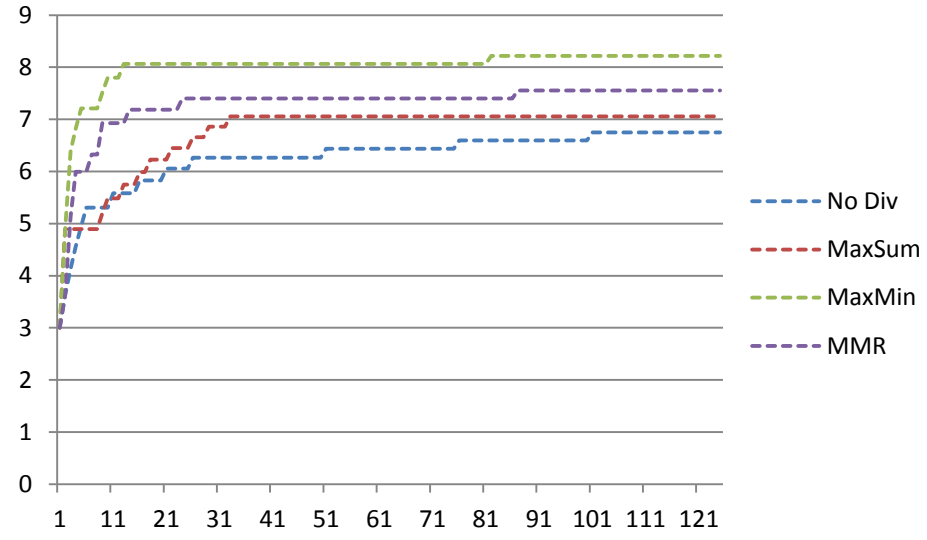


Figure 6-31 α-DCG Quantitative Diversification (α = 1.0, λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance and $\alpha - DCG_k$ has greatest novelty compare to relevance than MMR perform an excellent results and MaxMin perform very less than MMR but very good from the base line while MaxSum has also a good performance than the base line but less than MaxSum and MMR.

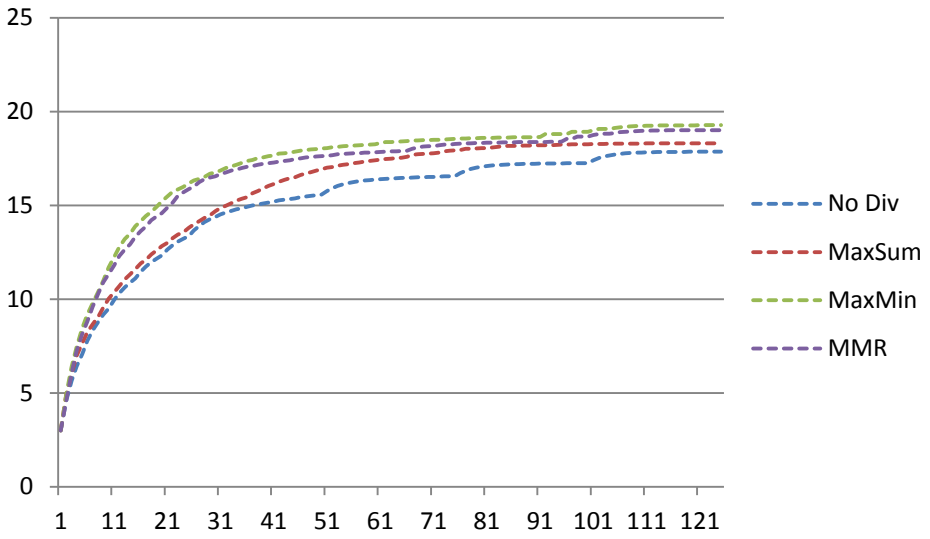


Figure 6-32 α-DCG Quantitative Diversification (α = 0.25, λ = 0.75)

When algorithm's diversification coefficient has greater value compare to relevance and $\alpha - DCG_k$ novelty element has a low value compare to relevance than MMR MaxMin, and MaxSum perform a little bit good from the base line.

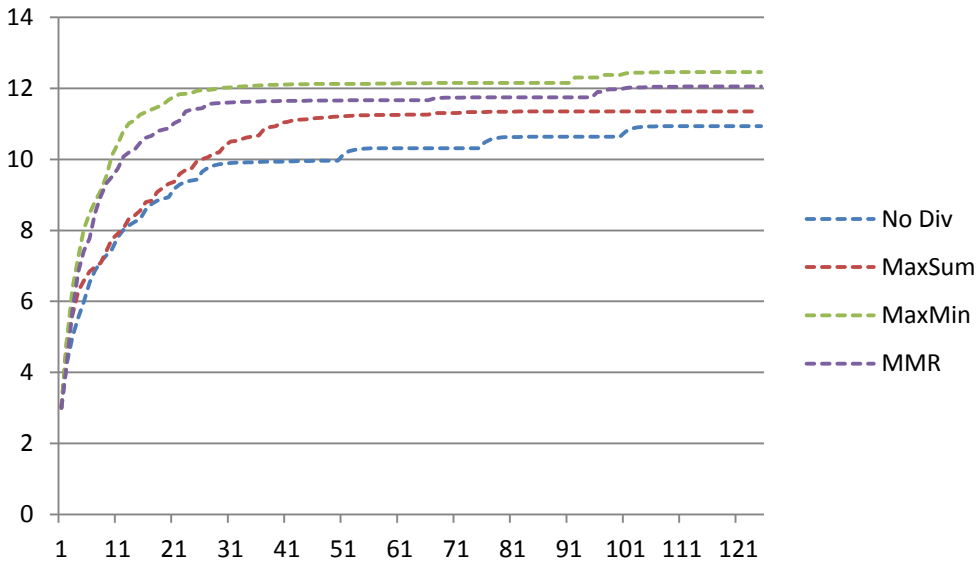


Figure 6-33 α-DCG Quantitative Diversification (α = 0.5, λ = 0.75)

When algorithm's diversification coefficient has greater value compare to relevance and $\alpha - DCG_k$ has equally novelty and relevance than MaxMin, MMR gives a good result from the base line while MaxSum has also a good performance than the base line but less than MaxSum and MMR.

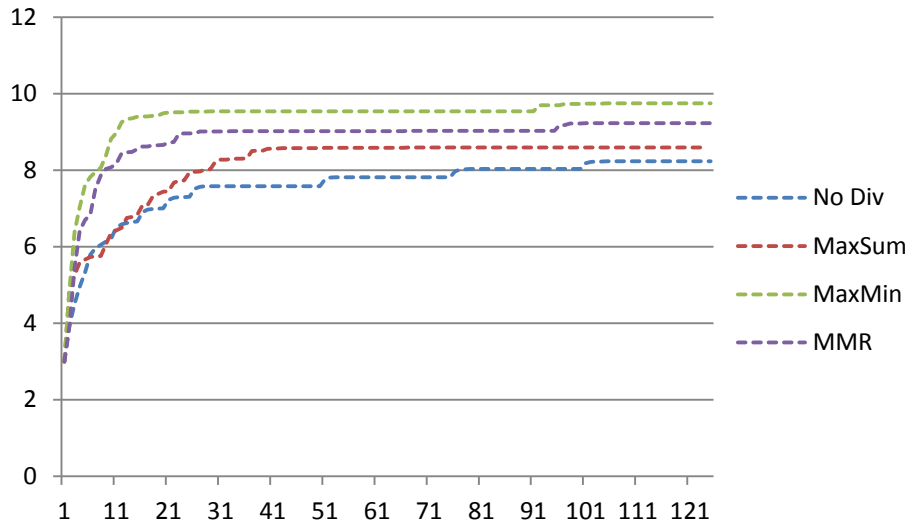


Figure 6-34 α-DCG Quantitative Diversification (α = 0.75, λ = 0.75)

When algorithm's diversification coefficient has greater value compare to relevance and $\alpha - DCG_k$ has greater novelty compare to relevance than MMR perform a very good results and MaxMin perform less than MMR but good from the base line while MaxSum has also a good performance than the base line but less than MaxSum and MMR

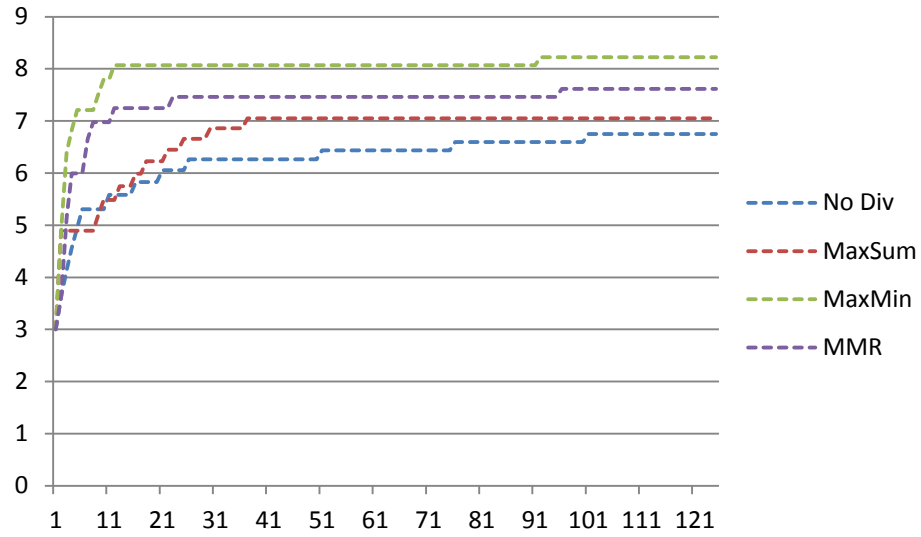


Figure 6-35 α-DCG Quantitative Diversification (α = 1.0, λ = 0.75)

When algorithm's diversification coefficient has greater value compare to relevance and $\alpha - DCG_k$ has greatest novelty compare to relevance than MMR perform an excellent results and MaxMin perform very less than MMR but very good from the base line while MaxSum has also a good performance than the base line but less than MaxSum and MMR

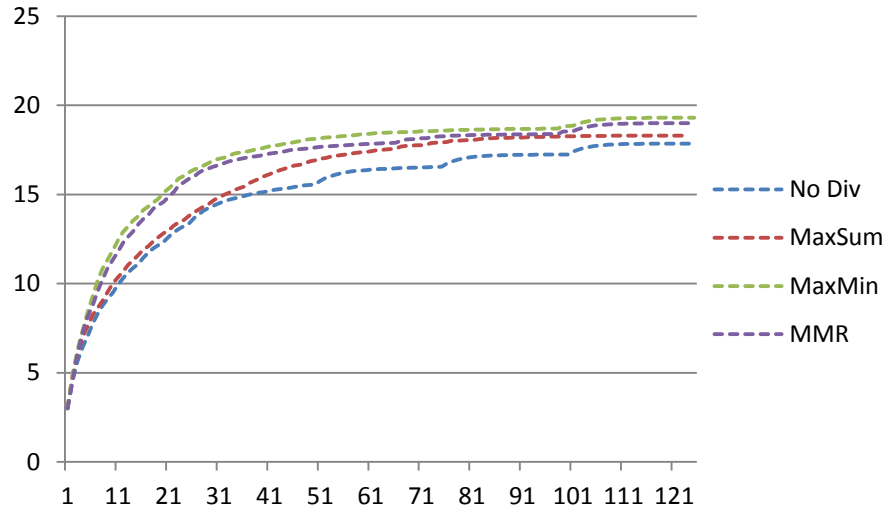


Figure 6-36 α-DCG Quantitative Diversification (α = 0.25, λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance and $\alpha - DCG_k$ novelty element has a low value compare to relevance than MMR MaxMin, and MaxSum perform a little bit good from the base line.

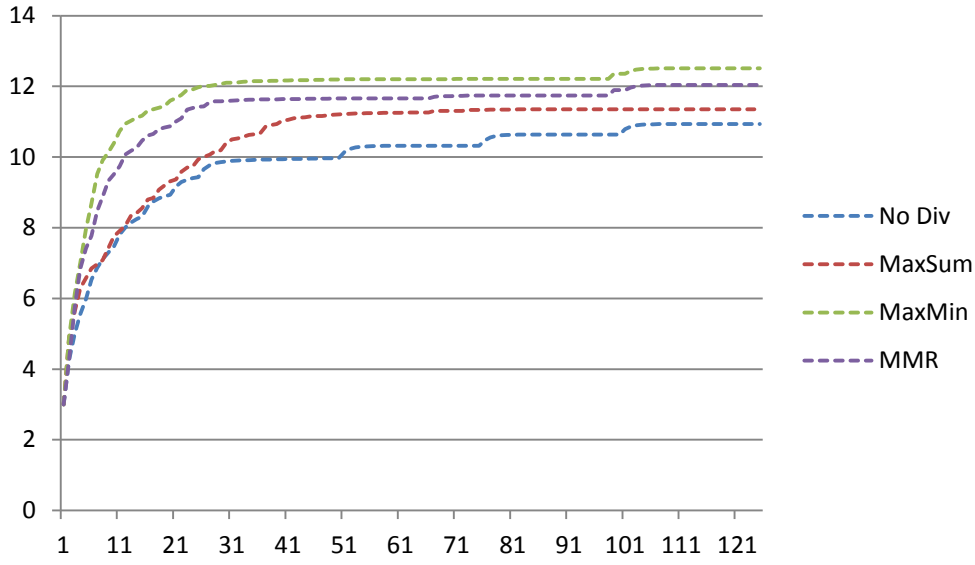


Figure 6-37 α-DCG Quantitative Diversification (α = 0.5, λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance and $\alpha - DCG_k$ has equally novelty and relevance than MaxMin, MMR gives a good result from the base line while MaxSum perform better than the base but less than MaxSum and MMR

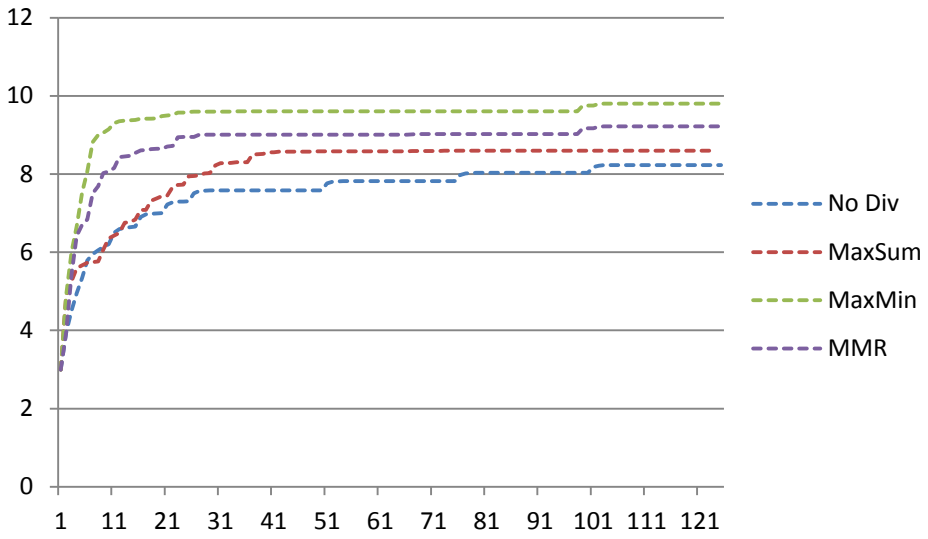


Figure 6-38 α-DCG Quantitative Diversification (α = 0.75, λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance and $\alpha - DCG_k$ has greater novelty compare to relevance than MMR perform a very good results and MaxMin perform less than MMR but good from the base line while MaxSum has a very little performance compare to the base line and MMR, MaxMin.

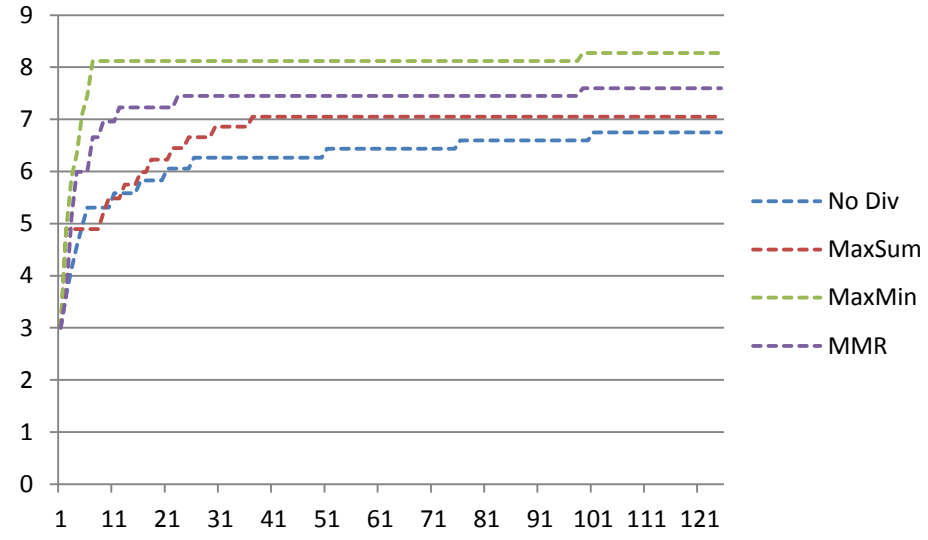


Figure 6-39 α-DCG Quantitative Diversification (α = 1.0, λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance and $\alpha - DCG_k$ has greatest novelty compare to relevance than MMR perform an excellent results and MaxMin perform very less than MMR but very good from the base line while MaxSum has a very little performance compare to the base line.

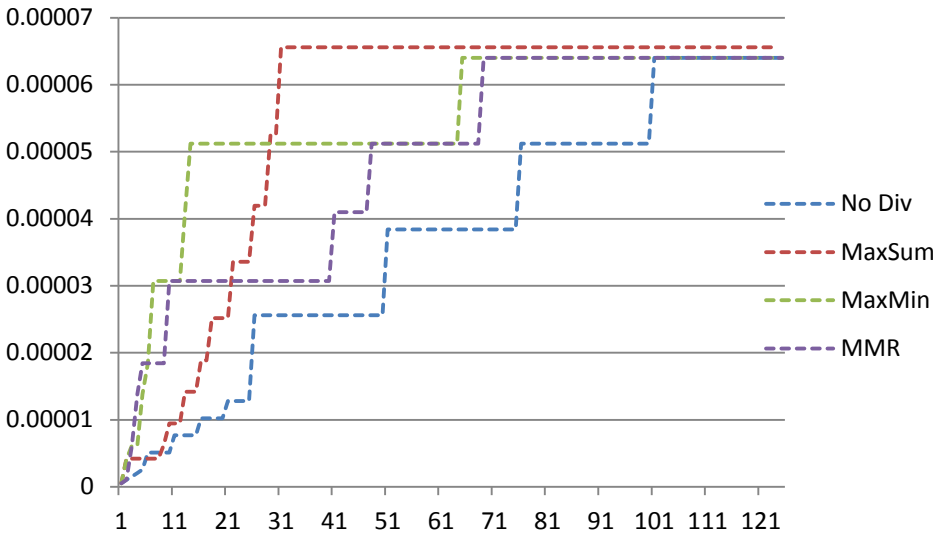


Figure 6-40 MD-Recall Quantitative Diversification (λ = 0.25)

When algorithm's diversification coefficient has a lower value than all algorithms (MaxMin, MMR, and MaxSum) gives good and almost equal performance from the base line.

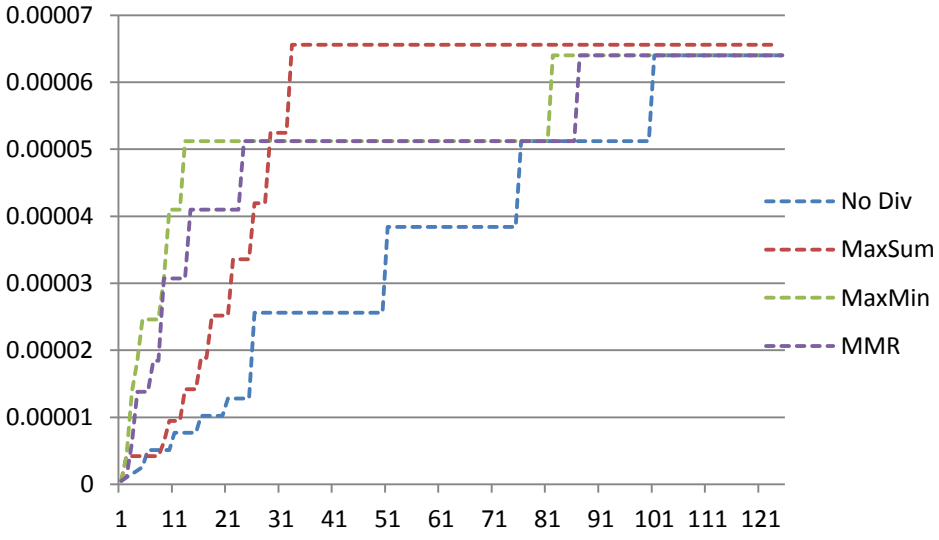


Figure 6-41 MD-Recall Quantitative Diversification (λ = 0.5)

When algorithm's diversification coefficient has equal value as of relevance than all algorithms (MaxMin, MMR, and MaxSum) gives good and almost equal performance from the base line.

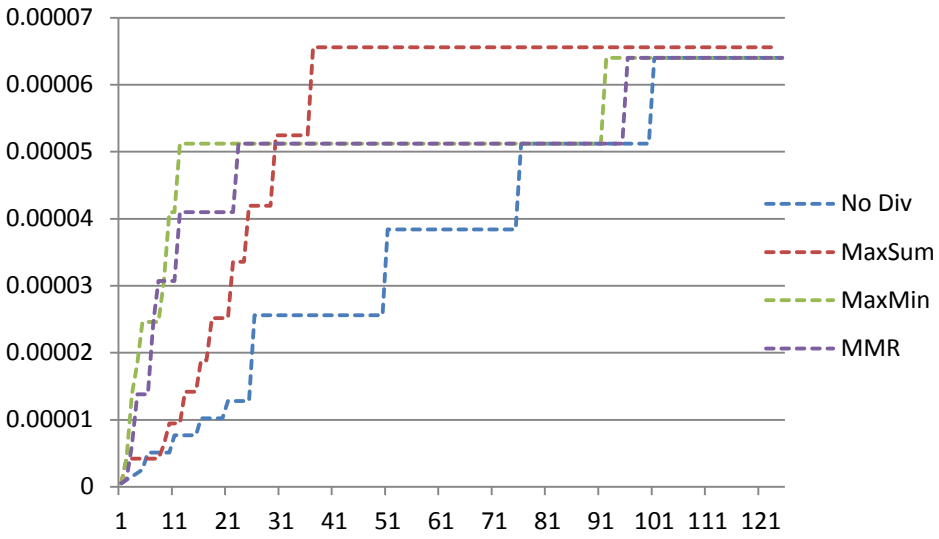


Figure 6-42 MD-Recall Quantitative Diversification (λ = 0.75)

When algorithm's diversification coefficient has greater value as of relevance than all algorithms (MaxMin, MMR, and MaxSum) gives good and almost equal performance from the base line.

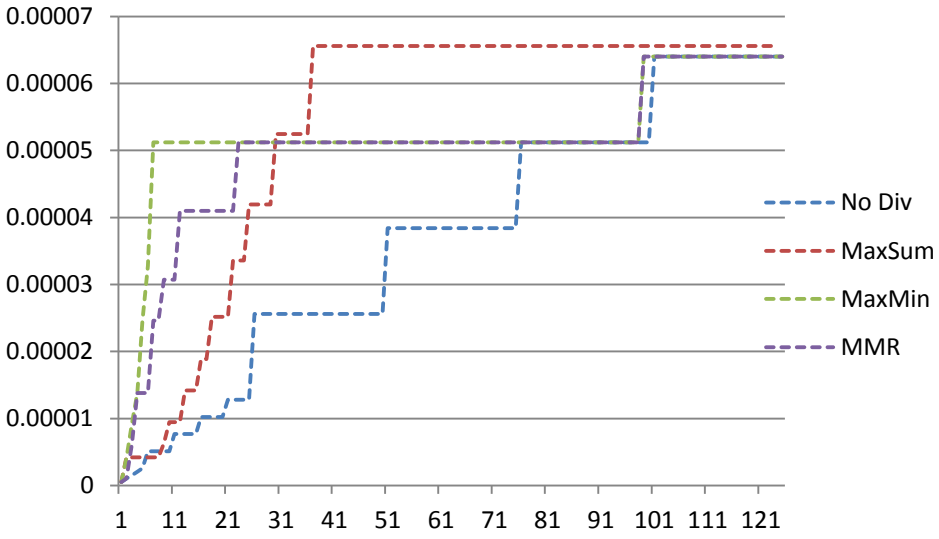


Figure 6-43 MD-Recall Quantitative Diversification (λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance than all algorithms (MaxMin, MMR, and MaxSum) gives good and almost equal performance from the base line.

6.1.6 Categorical and Quantitative Diversification

Suppose we want categorical and quantitative diversification based on hotel name, hotel lowest price, restaurant name, restaurant average price, museum name, and museum full fee. Then our semantic rule can be like this:

```
[
	{"semanticType": "SimpleDiff",
		"attributeName":["lowestPrice"],
		"diversificationWeight":0.20,
		"schemaId": "9Mip3B1IpOnFHJPZNJD3CH.1",
		"diversificationType":"quantitative"
	},{"semanticType": "SimpleDiff",
		"attributeName":["avgPrice"],
		"diversificationWeight":0.20,
		"schemaId": "GjvxAw2xhlnTPEx33A6SeK.2",
		"diversificationType":"quantitative"
	},{"semanticType": "SimpleDiff",
		"attributeName":["fullFee"],
		"diversificationWeight":0.20,
		"schemaId": "1X9oodXQQKlahX2pYO8P7g.3",
		"diversificationType":"quantitative"
	},{"semanticType": "SimpleDiff",
		"attributeName":["name"],
		"diversificationWeight":0.10,
		"schemaId": "9Mip3B1IpOnFHJPZNJD3CH.1",
		"diversificationType":"categorical"
	},{"semanticType": "SimpleDiff",
		"attributeName":["name"],
		"diversificationWeight":0.10,
		"schemaId": "GjvxAw2xhlnTPEx33A6SeK.2",
		"diversificationType":"categorical"
	},{"semanticType": "SimpleDiff",
		"attributeName":["name"],
		"diversificationWeight":0.20,
		"schemaId": "1X9oodXQQKlahX2pYO8P7g.3",
		"diversificationType":"categorical"
	}
]
```

Figure 6-44 Example Categorical and Quantitative Diversification JSON

$\alpha - DCG_k$ and $MD - Recall_k$ for the result sets obtained with no diversification and with the diversification algorithms MMR, MaxSum, and MaxMin, applying both categorical and quantitative distances. Each data point

of the X-axis represents the k-th element in the result-set. The Y-axes represent, respectively, the values of $\alpha - DCG_k$ and $MD - Recall_k$

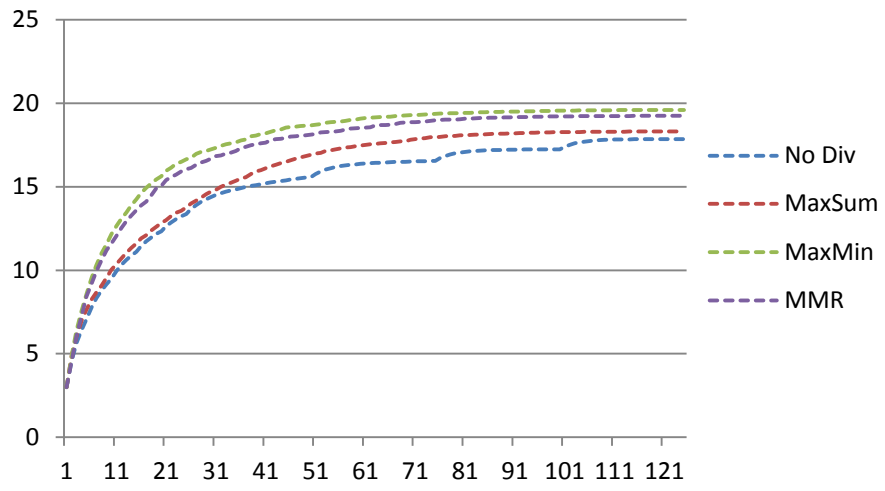


Figure 6-45 α-DCG Mix Diversification (α = 0.25, λ = 0.25)

When algorithm's diversification coefficient has a lower value and $\alpha - DCG_k$ $novelty\ element\ has\ a\ low$ value compare to relevance than MMR MaxMin, gives the good performance from the base line, while MaxSum has a little performance.

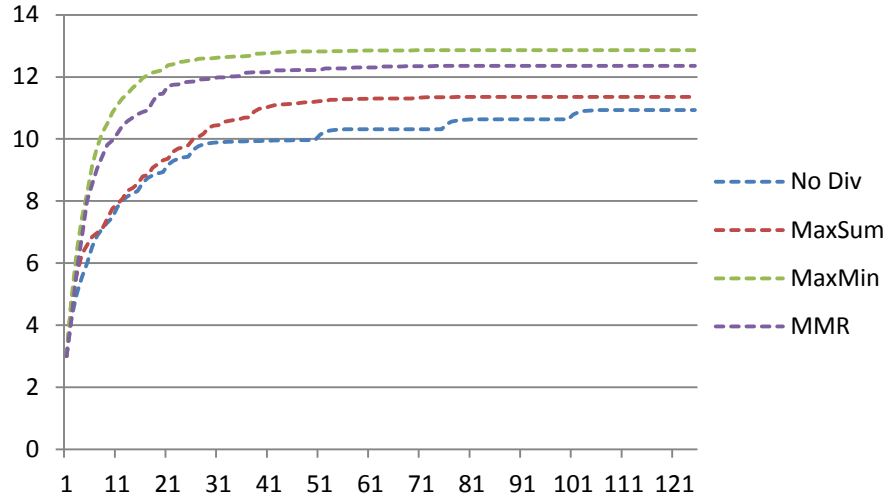


Figure 6-46 α-DCG Mix Diversification (α = 0.5, λ = 0.25)

When algorithm's diversification coefficient has a lower value and $\alpha - DCG_k$ has equally novelty and relevance than MaxMin and MMR gives a good result from the base line and almost equal while MaxSum perform a little bit good from the base line.

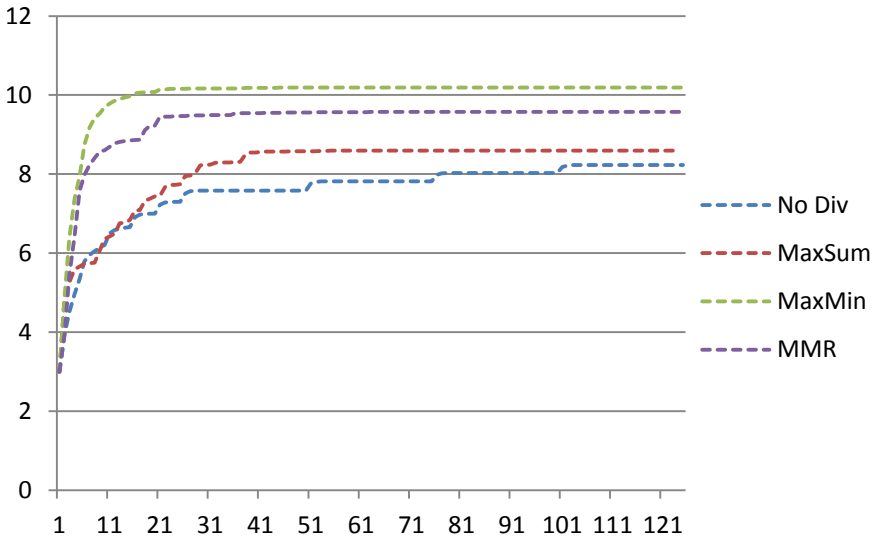


Figure 6-47 α-DCG Mix Diversification (α = 0.75, λ = 0.25)

When algorithm's diversification coefficient has a lower value and $\alpha - DCG_k$ has greater novelty compare to relevance than MMR perform very good results, MaxMin perform less then MMR but good from the base line while MaxSum has a very little improvement compare to the base line.

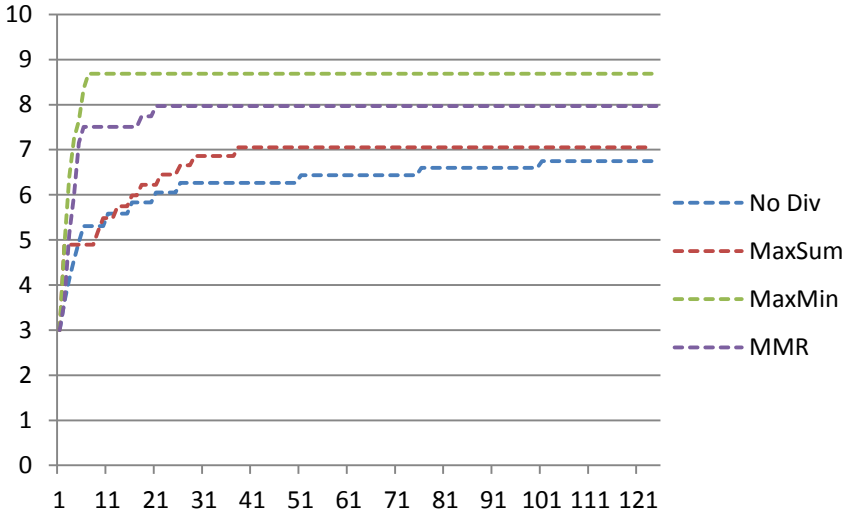


Figure 6-48 α-DCG Mix Diversification (α = 1.0, λ = 0.25)

When algorithm's diversification coefficient has a lower value and $\alpha - DCG_k$ has greatest novelty compare to relevance than MMR perform a very good results and MaxMin perform less than MMR but good from the base line while MaxSum has a very little improvement compare to the base line.

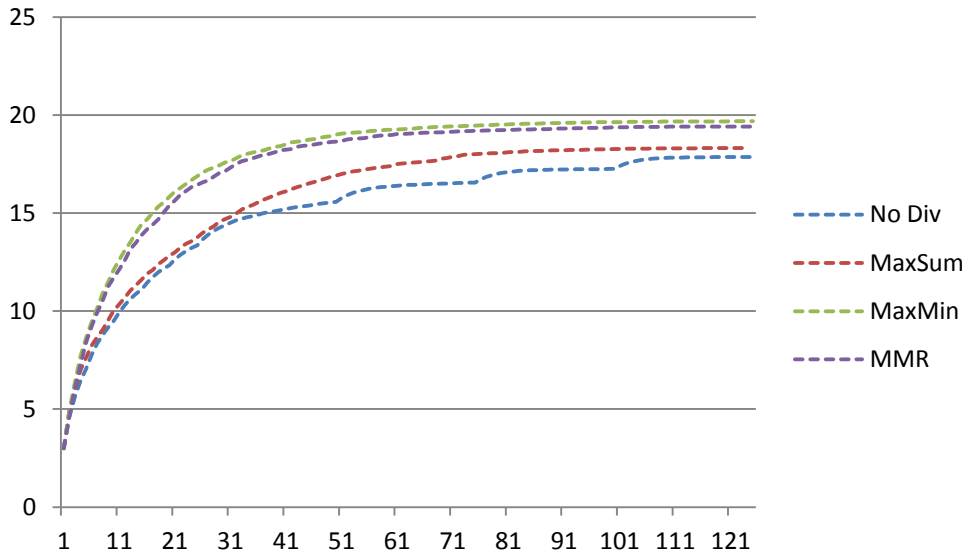


Figure 6-49 α-DCG Mix Diversification (α = 0.25, λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance and $\alpha - DCG_k$ novelty element has a low value compare to relevance than MMR, MaxMin gives good performance from the base line, while MaxSum has a very short performance.

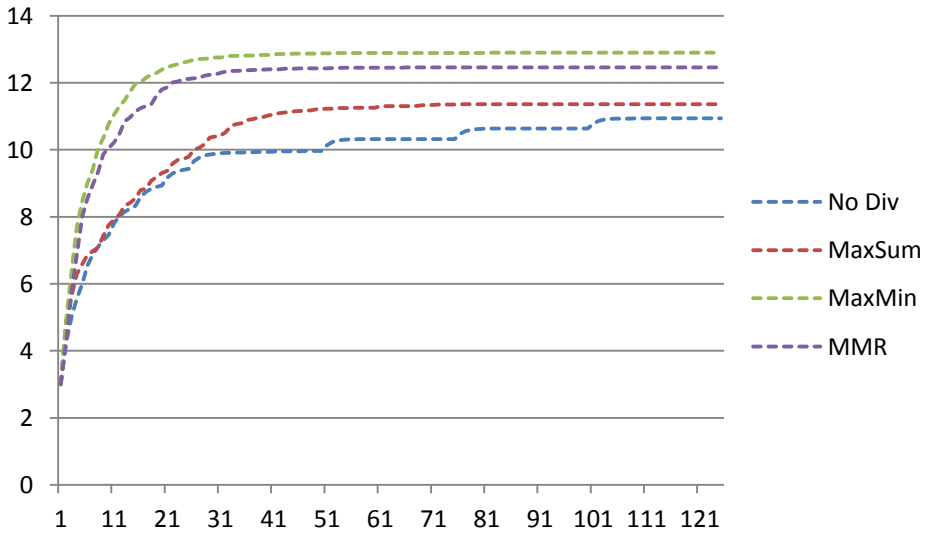


Figure 6-50 α-DCG Mix Diversification (α = 0.5, λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance and $\alpha - DCG_k$ has equally novelty and relevance than MaxMin and MMR gives a good result from the base line and almost equal while MaxSum perform a bit good from the base line.

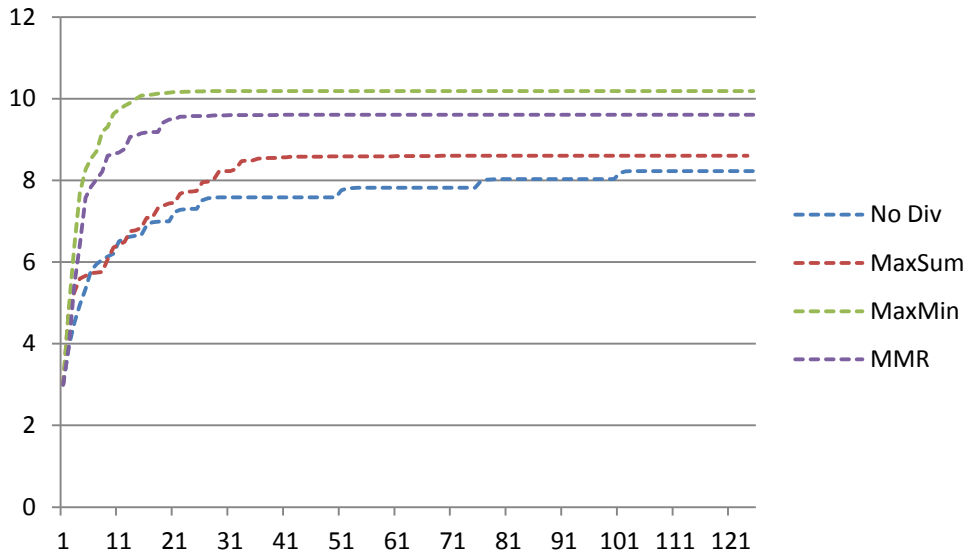


Figure 6-51 α-DCG Mix Diversification (α = 0.75, λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance and $\alpha - DCG_k$ has greater novelty compare to relevance than MaxMin and MMR gives a good result from the base line and almost equal while MaxSum perform a bit good from the base line.

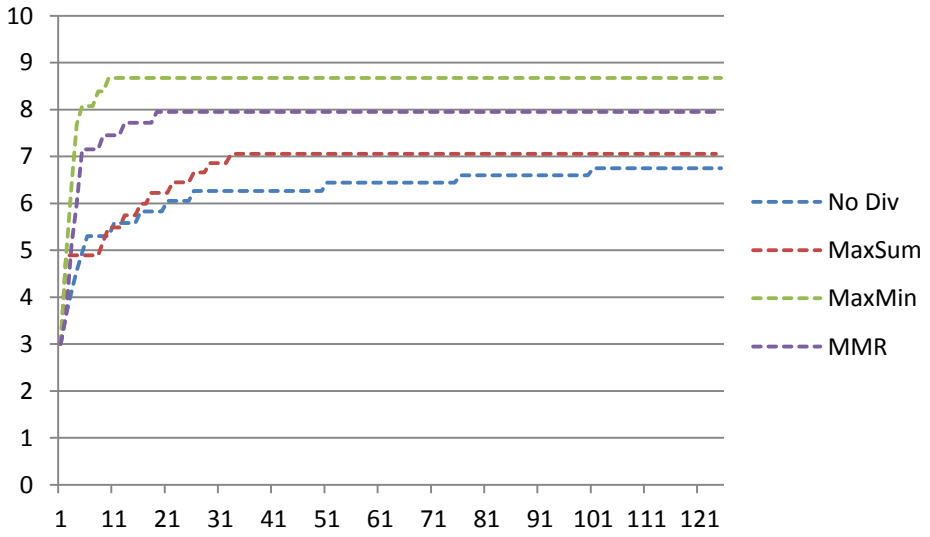


Figure 6-52 α-DCG Mix Diversification (α = 1.0, λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance and $\alpha - DCG_k$ has greatest novelty compare to relevance than MaxMin perform a very good results and MMR perform less than MaxMin but good from the base line while MaxSum has a very little improvement compare to the base line.

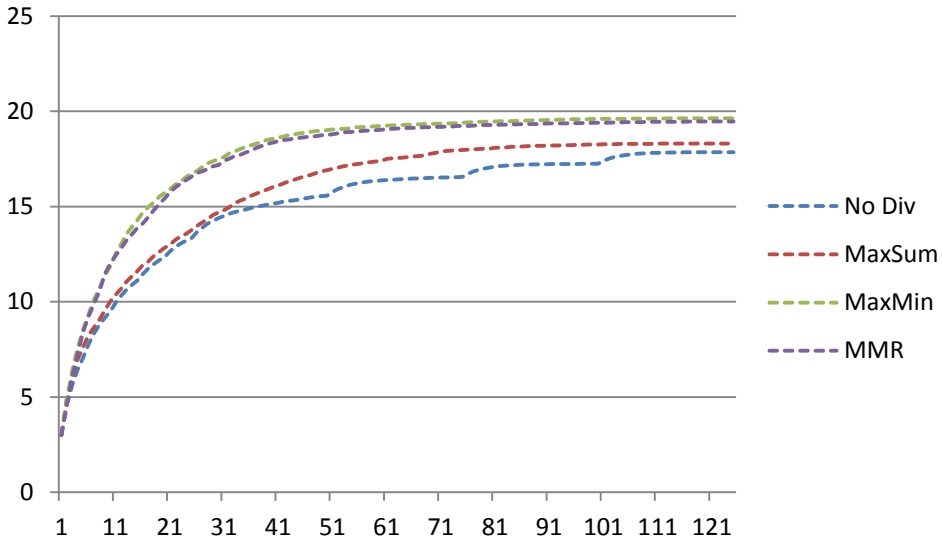


Figure 6-53 α-DCG Mix Diversification (α = 0.25, λ = 0.75)

When algorithm's diversification coefficient has greater value compare to relevance and $\alpha - DCG_k$ novelty element has a low value compare to relevance than MMR, MaxMin gives good performance from the base line, while MaxSum has a very short performance.

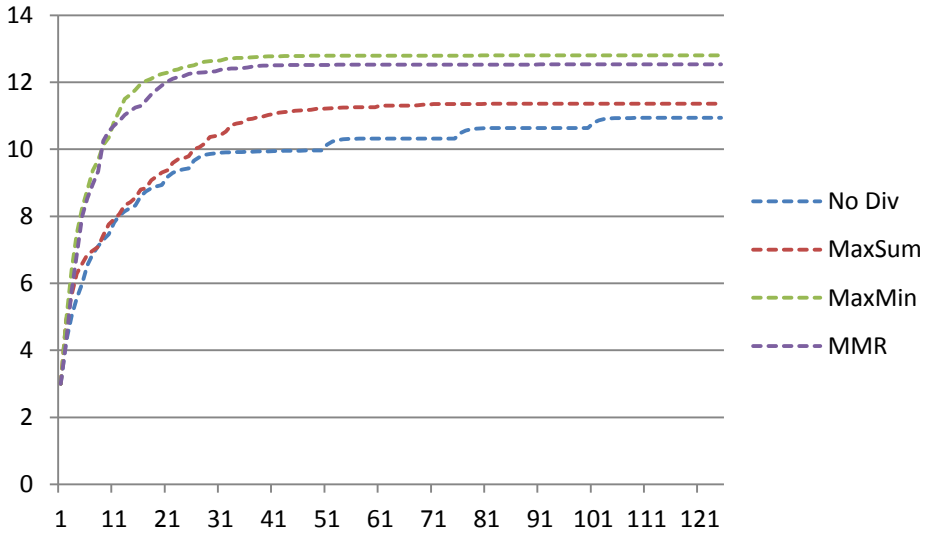


Figure 6-54 α-DCG Mix Diversification (α = 0.5, λ = 0.75)

When algorithm's diversification coefficient has greater value compare to relevance and $\alpha - DCG_k$ has equally novelty and relevance than MaxMin and MMR gives a good result from the base line and has negligible difference with each other while MaxSum perform a little improvement from the base line.

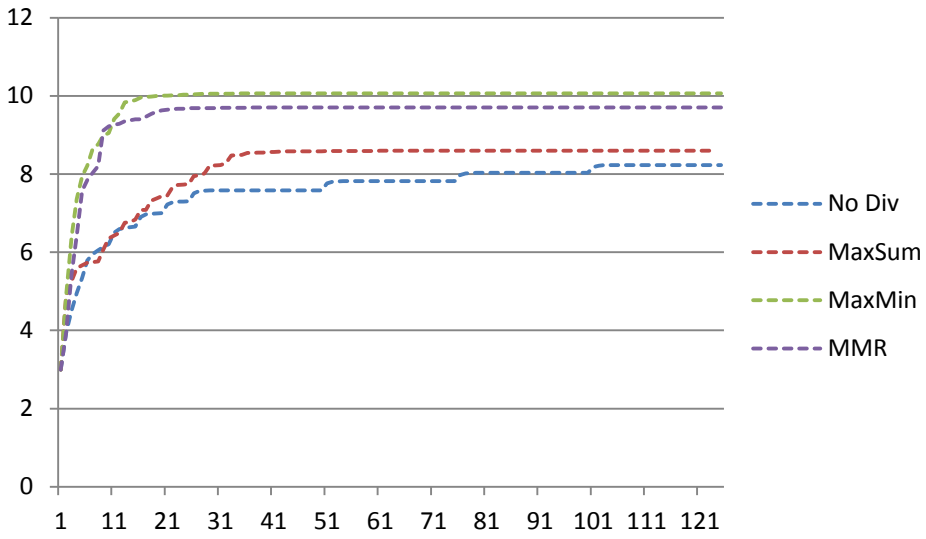


Figure 6-55 α-DCG Mix Diversification (α = 0.75, λ = 0.75)

When algorithm's diversification coefficient has greater value compare to relevance and $\alpha - DCG_k$ has greater novelty compare to relevance than MaxMin and MMR gives a good result from the base line and almost equal while MaxSum perform a little improvement from the base line.

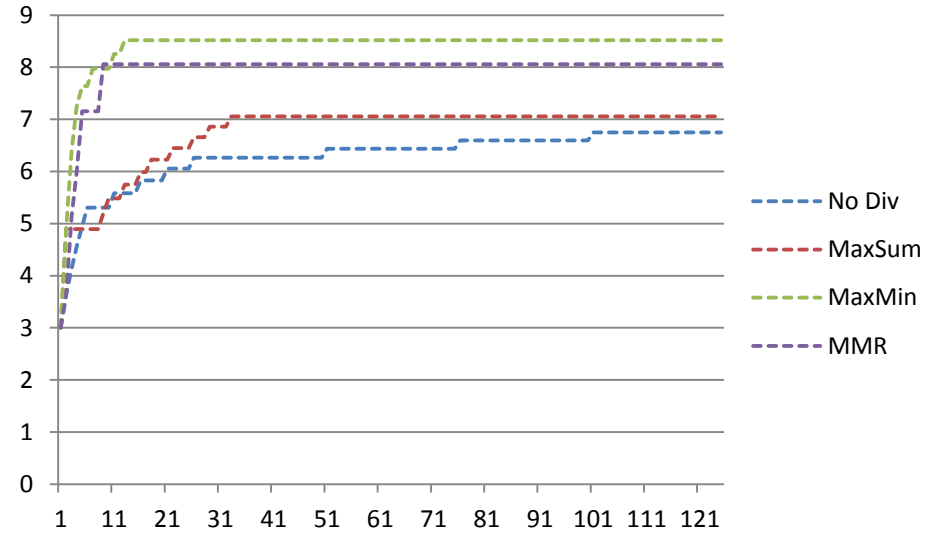


Figure 6-56 α-DCG Mix Diversification (α = 1.0, λ = 0.75)

When algorithm's diversification coefficient has greater value compare to relevance and $\alpha - DCG_k$ has greatest novelty compare to relevance than MMR perform a very good results and MaxMin perform less than MMR but good from the base line while MaxSum has a very little improvement compare to the base line.

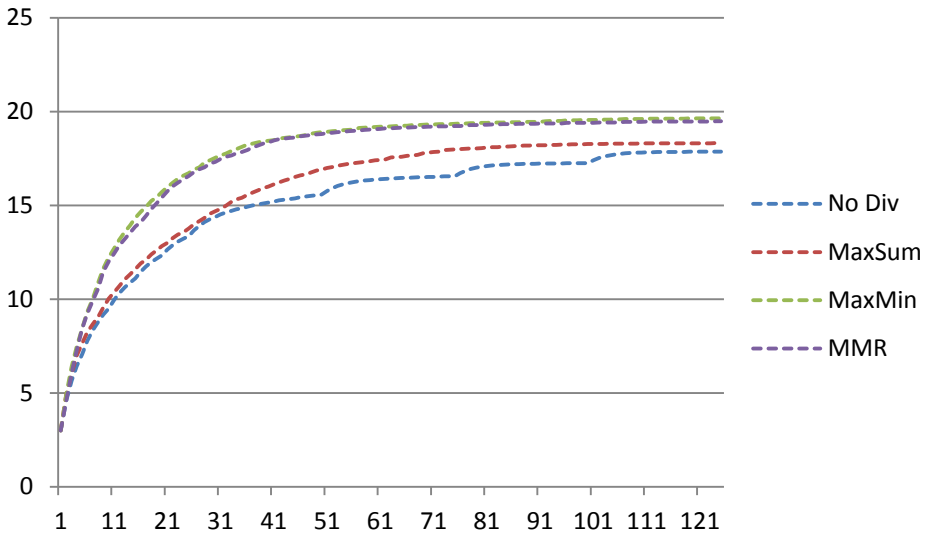


Figure 6-57 α-DCG Mix Diversification (α = 0.25, λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance and $\alpha - DCG_k$ novelty element has a low value compare to relevance than MMR MaxMin, gives the good performance from the base line, while MaxSum has a very short performance.

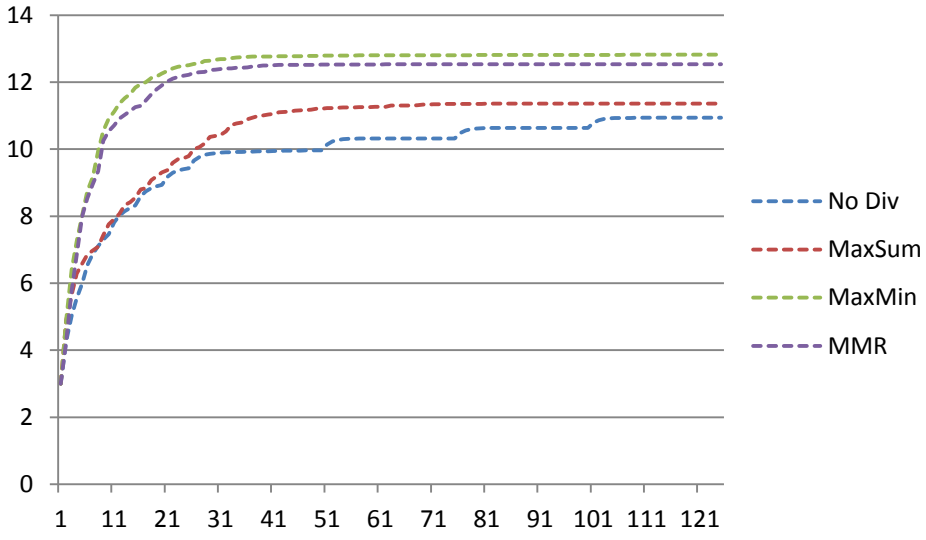


Figure 6-58 α-DCG Mix Diversification (α = 0.5, λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance and $\alpha - DCG_k$ has equally novelty and relevance than MaxMin and MMR gives a good result from the base line and has negligible difference with each other while MaxSum perform a little bit good from the base line.

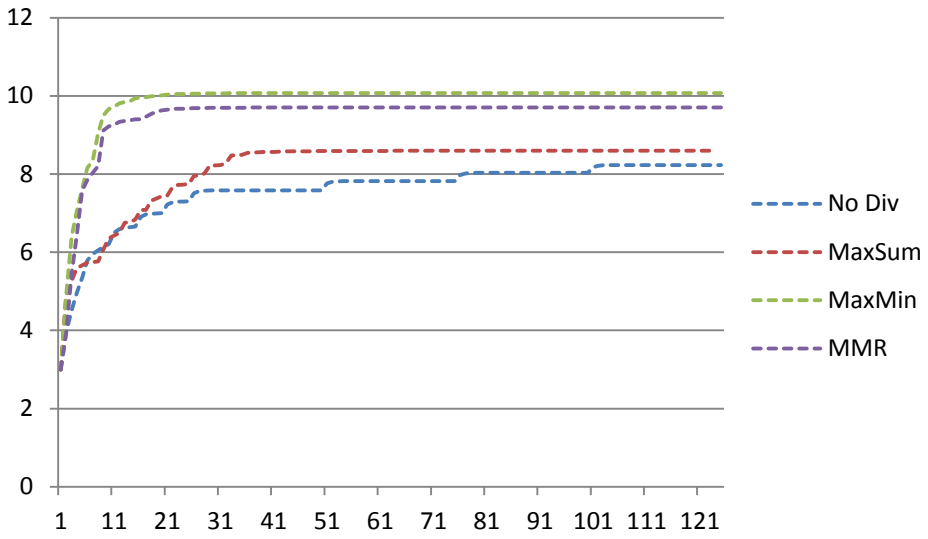


Figure 6-59 α-DCG Mix Diversification (α = 0.75, λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance and $\alpha - DCG_k$ has greater novelty compare to relevance than MMR perform very good results, MaxMin perform less then MMR but good from the base line while MaxSum has a very little improvement compare to the base line.

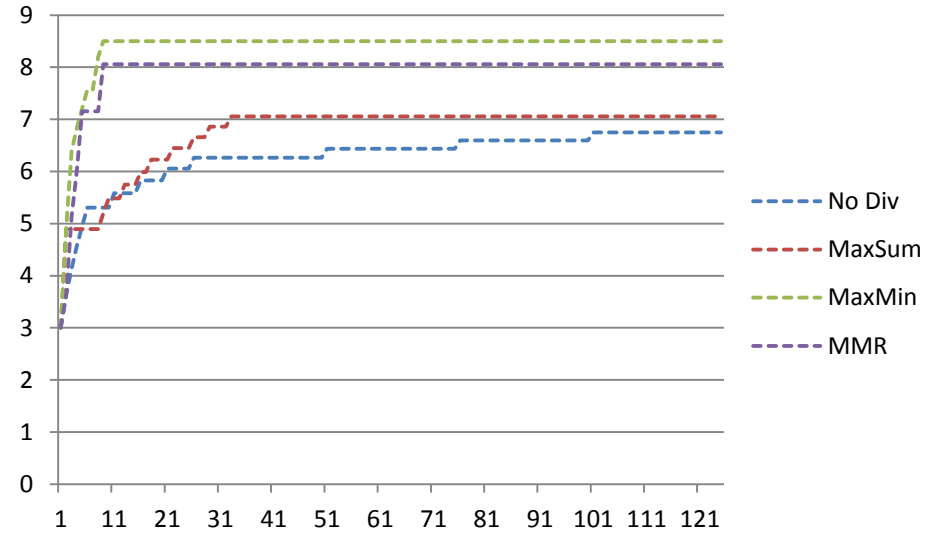


Figure 6-60 α-DCG Mix Diversification (α = 1.0, λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance and $\alpha - DCG_k$ has greatest novelty compare to relevance than MMR perform a very good results and MaxMin perform less than MMR but good from the base line while MaxSum has a very little improvement compare to the base line.

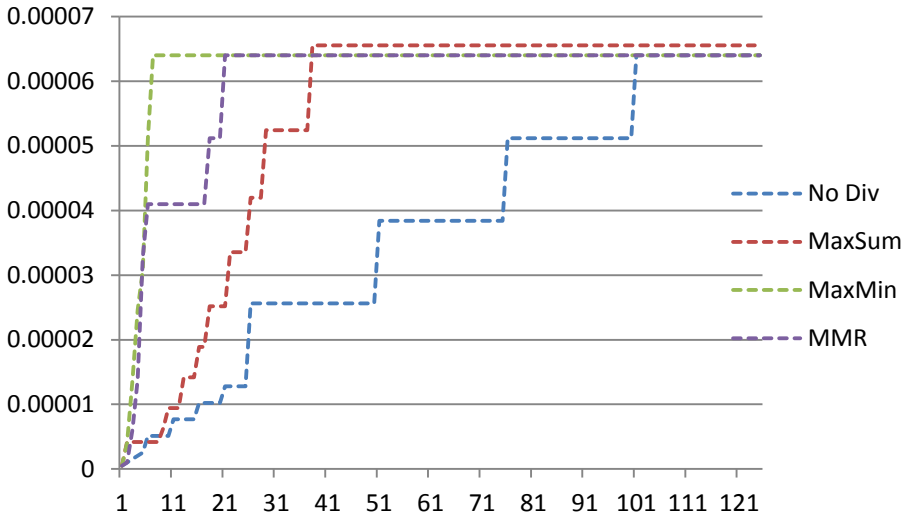


Figure 6-61 MD-Recall Mix Diversification (λ = 0.25)

When algorithm's diversification coefficient has a lower value then MaxMin, and MMR algorithms give a good and almost equal performance from the base line, while MaxSum give less performance than MaxMin and MMR but good from the base line.

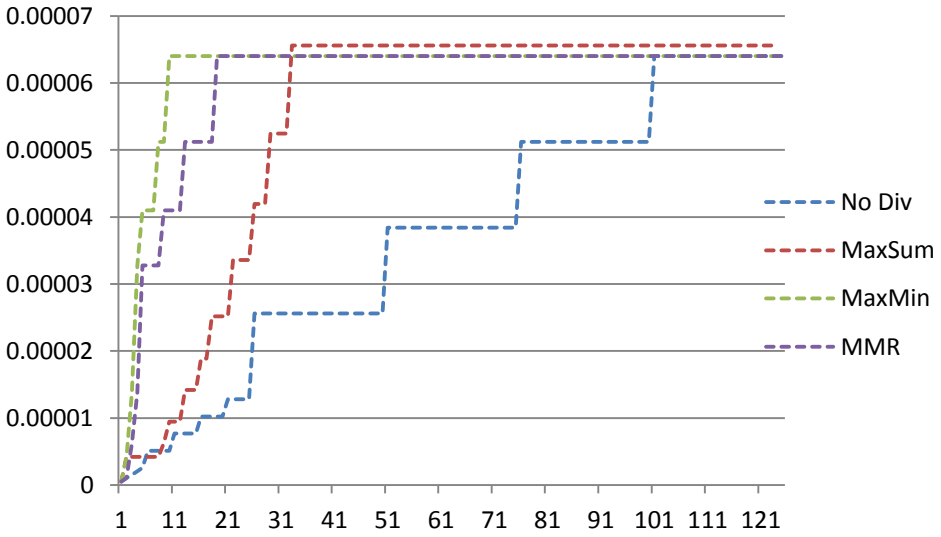


Figure 6-62 MD-Recall Mix Diversification (λ = 0.5)

When algorithm's diversification coefficient has equal value as the relevance than MaxMin, and MMR algorithms give a good and almost equal performance from the base line, while MaxSum give less performance than MaxMin and MMR but good from the base line.

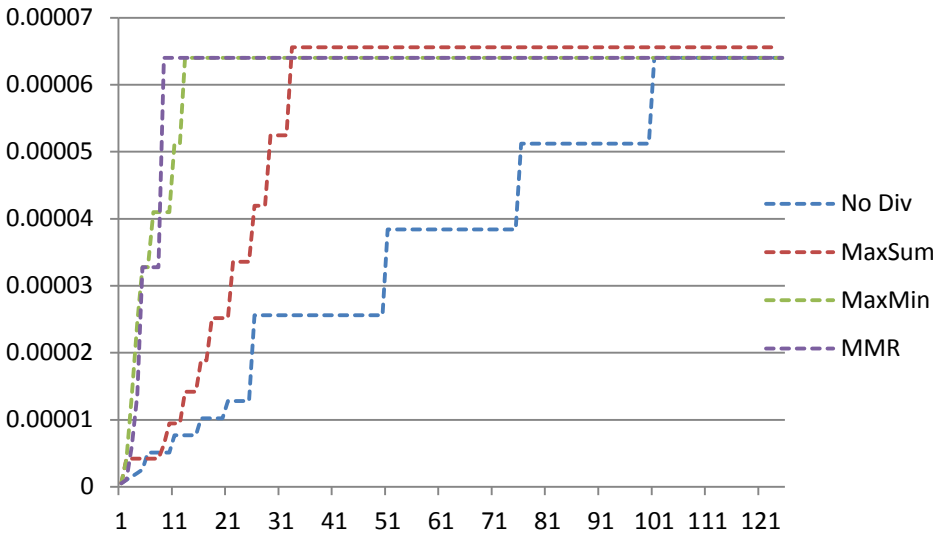


Figure 6-63 MD-Recall Mix Diversification (λ = 0.75)

When algorithm's diversification coefficient has greater value compare to relevance than MaxMin, and MMR algorithms give a good and almost equal performance from the base line, while MaxSum give less performance than MaxMin and MMR but good from the base line.

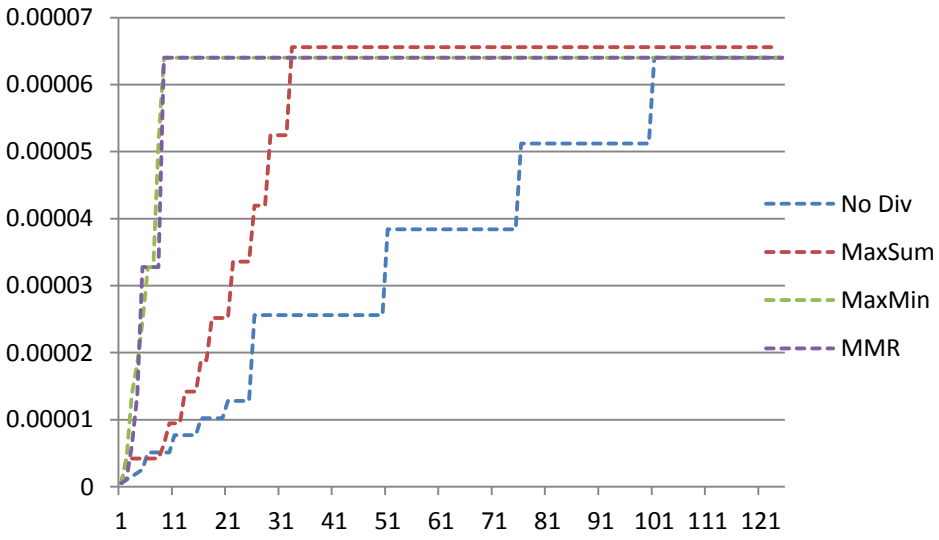


Figure 6-64 MD-Recall Mix Diversification (λ = 1.0)

When algorithm's diversification coefficient has greatest value compare to relevance than MaxMin, and MMR algorithms give a good and almost equal performance from the base line, while MaxSum give less performance than MaxMin and MMR but good from the base line.

6.2 Complexity

Recall the pseudo code section of the implementation chapter.

Suppose:

Total number of the data tuples = n
Total number of Semantic Rules = s
Maximum number of attribute in a Semantic Rule = a
Number of the element that we want in the Result Set = K

6.2.1 Building RelevantDiverseStructure Array

Complexity to iterate for each and every possible combination of data key is given by:

$$= O((n-1) + (n-2) + \ldots + (1))$$

$$= O\left(\frac{n * (n-1)}{2}\right)$$

$$= O\left(\frac{n^2 - n}{2}\right)$$

$$= \frac{1}{2} O(n^2 - n)$$

$$= O(n^2 - n)$$

$$= O(n^2) - O(n)$$

$$= O(n^2)$$

As each combination is needed to iterate on each and every semantic rule for the desire output, so complexity becomes

$$= O(n^2 s)$$

We know that each semantic rule has maximum $\boldsymbol{a}$ attributes, so that final complexity becomes

$$= O(n^2\ s\ a)$$

6.2.2 Get Normalized Diversification Score

Complexity to find out total normalized score from an object of RelevantDiverseStructure is given below:

$$\text{Total Complexity} = O(s)$$

6.2.3 MaxSum

As we already know cost of building the data RelevantDiverseStructure complexity i.e.

$$= O(n^2\ s\ a)$$

Cost for getting the Normalized Diversify score

$$= O(s)$$

So, Total Complexity:

$$= O(n^2\ s\ a) + O\left(\frac{K * \left((n^2 s + n^2)\right)}{2}\right)$$

$$= O(n^2\ s\ a) + O\left(\frac{K * \left((n^2 s)\right)}{2}\right) + O\left(\frac{K * \left((n^2)\right)}{2}\right)$$

$$= O(n^2\ s\ a) + O\left(\frac{K * (n^2 s)}{2}\right)$$

$$= O(n^2\ s\ a) + \frac{1}{2} O\left(K * (n^2 s)\right)$$

$$= O(n^2\ s\ a) + O(K n^2 s)$$

$$= O(n^2 s\ a) + O(n^2 s\ K)$$

We know that in normal cases maximum number of attribute in a semantic rule is less than the number of data tuple that we want in the result set. Mathematically

$$a < K$$

So from the above equation we can also write total complexity

$$= \boldsymbol{O(K n^2)}$$

If user only want to see the diversification but not want to filter the tuple than

$$K = n$$

and the total complexity becomes

$$= O(n^3)$$

6.2.4 Get RelevantDiverseStructure

As we know that the total number of combination is

$$n(n-1)$$

So to search in $n(n-1)$ object we have to iterate all of these combination, which mean that the total complexity is:

$$= O\big(n(n-1)\big)$$

$$= O(n^2 + n)$$

$$= O(n^2) + O(n)$$

$$= O(n^2)$$

6.2.5 MaxMin

As we already know cost of building the data RelevantDiverseStructure complexity i.e.

$$= O(n^2\ s\ a)$$

Cost for getting the Normalized Diversify score

$$= O(s)$$

Cost of getting a combination that has dataKeyU and dataKeyV

$$= O(n^2)$$

So, Total Complexity:

$$= O(n^2\ s\ a) + O(n^2\ s\ ) + O\left(K\left(n(1+2+3+\cdots K-1) + ns(1n^2+2n^2+3n^2+\cdots(K-1)n^2)\right)\right)$$

$$= O(n^2\ s\ a) + O\left(K\ n(1+2+3+\cdots K-1)\right) + O\left(Kn^3s(1+2+3+\cdots K)\right)$$

$$= O(n^2\ s\ a) + O\left(K\ n^3s(1+2+3+\cdots K)\right)$$

$$= O(n^2\ s\ a) + O\left(Kn^3s\left(\frac{K(K+1)}{2}\right)\right)$$

$$= O(n^2\ s\ a) + \frac{1}{2}O\left(K\ n^3s(K^2+\ K)\right)$$

$$= O(n^2\ s\ a) + O(K^3n^3s) + O(K^2ns)$$

$$= O(n^2\ s\ a) + O(K^3n^3s)$$

$$= \boldsymbol{O(n^2\ s\ a) + O(n^3s\ K^3)}$$

If user only want to see the diversification but not want to filter the tuple than

$$K = n$$

So,

$$= O(n^2\ s\ a) + O(n^3s\ n^3)$$

$$= O(n^2\ s\ a) + O(n^2 s\ n^4)$$

We know that in normal cases maximum number of attribute in a semantic rule is much less than the double square of the result set number of data tuple, mathematically

$$a \ll n^4$$

So from the above equation we can also write total complexity

$$= O(n^2 s\ n^4)$$

$$= O(n^6 s\ )$$

6.2.6 MMR

As we already know cost of building the data RelevantDiverseStructure complexity i.e.

$$= O(n^2\ s\ a)$$

Cost for getting the Normalized Diversify score

$$= O(s)$$

Cost of getting a combination that has dataKeyU and dataKeyV

$$= O(n^2)$$

So, Total Complexity:

$$= O(n^2\ s\ a) + O(n^2) + O\left(K\big(n(1+2+3+\cdots K-1) + ns(1n^2+2n^2+3n^2+\cdots(K-1)n^2)\big)\right)$$

$$= O(n^2\ s\ a) + O\big(K\ n(1+2+3+\cdots K-1)\big) + O\big(Kn^3s(1+2+3+\cdots K)\big)$$

$$= O(n^2\ s\ a) + O\big(K\ n^3s(1+2+3+\cdots K)\big)$$

$$= O(n^2\ s\ a) + O\left(Kn^3s\left(\frac{K(K+1)}{2}\right)\right)$$

$$= O(n^2\ s\ a) + \frac{1}{2}O\big(K\ n^3s(K^2+K)\big)$$

$$= O(n^2\ s\ a) + O(K^3n^3s) + O(K^2ns)$$

$$= O(n^2\ s\ a) + O(K^3n^3s)$$

$$= \boldsymbol{O(n^2\ s\ a) + O(n^3s\ K^3)}$$

If user only want to see the diversification but not want to filter the tuple than

$$K = n$$

So,

$$= O(n^2\ s\ a) + O(n^3s\ n^3)$$

$$= O(n^2\ s\ a) + O(n^2 s\ n^4)$$

We know that in normal cases maximum number of attribute in a semantic rule is much less than the double square of the result set number of data tuple, mathematically

$$a \ll n^4$$

So from the above equation we can also write total complexity

$$= O(n^2 s\ n^4)$$

$$= O(n^6 s\ )$$

Chapter 7. Conclusion & Future work

Multi-domain search is a promising trend in search applications, which can lead to the mash-up of multiple Webdata sources in ways that can serve many complex search processes today not well supported by search engines. To reap the benefits of search across different domains, the combinatorial explosion of result sets formed by several correlated entity instances must be tamed, to preserve the current capacity of search engines to squeeze in one page the most interesting results.

In this thesis, we have investigated the problem of computing a relevant and diverse set of correlated entity instances in response to a multi-domain query, by showing how the diversification techniques well studied in the context of IR can be extended to support this class of applications. Three algorithms for the trade-off between relevance and diversity have been tested experimentally, showing that they can introduce a significant degree of diversification in the result set.

In categorical diversification, MMR and MaxMin always outperform the un-diversified baseline when used with the categorical diversity function both in terms of $\alpha - DCG_k$ and $MD - Recall_k$; MaxSum instead does not provide significant improvements with respect to the baseline. One can also notice that MMR and MaxMin offer similar performance: this is not surprising, as the greedy algorithms for the two objective functions are also very similar.

In quantitative distance, instead, all algorithms provide similar performance. In particular MMR and MaxMin degrade their performance with respect to categorical distance, and behave only slightly better than the baseline. This may also be influenced by the chosen quality measures ($\alpha - DCG_k$ and $MD - Recall_k$), those are based on diversity of extracted objects and therefore are more suited to evaluate categorical diversification. Nonetheless, we notice an overall coherent behavior of MMR and MaxMin in all settings; likewise, MaxSum consistently performs similarly to the un-diversified case. Overall, these results support the hypothesis that diversification algorithms can improve the quality of multi-domain result sets.

In performance assessment, MaxSum has only $O(n^3)$ complexity and is much better than MaxMin and MMR which have $O(n^6 s)$.

Finally, we develop a careful graphical user interface design and a novel round of user testing of the multi-domain search concept.

In the future, we plan to extend this work in several directions. On the methodological side, there is the need of characterizing the interplay between the score function and the similarity measure, and to better understand the circumstances when relevance alone is sufficient to diversify, beyond the simple case of uniform data distribution studied in Section 2. The goal is to end up with a clear methodology for application developers to choose the most promising scoring and diversity functions, based on the application domain, available statistics on data distribution, and user preferences. On the architecture side, the prototype diversification component developed for the user study, which works off-line, is being integrated into an existing architecture for multi-domain search application development. This will bring about issues like the design of appropriate data and index structures for efficient diversification, relevance and diversity-aware caching of results, and the thorough evaluation of the overhead of diversification. One interesting issue is how to integrate diversification with the techniques currently implemented for optimizing rank-join queries over Web services with different response times and distributions of scores.

Chapter 8. Bibliography

1. **S. Ceri, M. Brambilla.** *Search Computing Challenges and Directions.* s.l. : Springer LNCS, March 2010. pp. 3-10. Vol. 5950.

2. *Diversification for multi-domain result sets.* **A. Bozzon, M. Brambilla, P. Fraternali, M. Tagliasacchi.** New York, NY, USA : ACM, 2011. CIKM '11 Proceedings of the 20th ACM international conference on Information and knowledge management .

3. *Supporting top-k join queries in relational databases.* **I. F. Ilyas, W. G. Aref, A. K. Elmagarmid.** 3, s.l. : Springer-Verlag 2004, August 2004, VLDB Journal, Vol. 13, pp. 207 - 221. 10.1007/s00778-004-0128-2.

4. *Building ranked mashups of unstructured sources with uncertain information.* **M. A. Soliman, I. F. Ilyas, M. Saleeb.** s.l. : PVLDB, 2010. Vol. 3(1), pp. 826 - 837.

5. *An axiomatic approach for result diversification.* **S. Gollapudi, A. Sharma.** New York, NY, USA : ACM, 2009. WWW '09: Proceedings of the 18th international conference on World wide web. pp. 381 - 390.

6. *Concept search: Semantics enabled syntactic search.* **F. Giunchiglia, U. Kharkevich, I. Zaihrayeu.** 2008. SemSearch. pp. 109 - 123.

7. *Kosmix: Exploring the deep web using Taxonomies and Categorization.* **A. Rajaraman.** 2009. PVLDB. Vol. 2(2), pp. 1524 - 1529.

8. *Constructing and exploring composite items.* **S. B. Roy, S. Amer-Yahia, A. Chawla, G. Das, C. Yu.** 2010. SIGMOD Conference. pp. 843 - 854.

9. *Search result diversification.* **M. Drosou, E. Pitoura.** 2010. SIGMOD Rec. Vol. 39(1), pp. 41-47.

10. *Current approaches to search result diversification.* **E. Minack, G. Demartini, W. Nejdl.** October 2009. First International Workshop on Living Web: Making Web Diversity a true asset. Vol. 515.

11. *Diversifying search results.* **R. Agrawal, S. Gollapudi, A. Halverson, S. Ieong.** New York, NY : ACM, 2009. WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining.

12. **N. Bansal, K. Jain, A. Kazeykina, J. Naor.** Approximation algorithms for diversification search ranking. [book auth.] C. Gavoille, C. Kirchner, S. Abramsky. *Automata, Languages and Programming.* Berlin / Heidelberg : Springer, 2010, Vol. 6199, pp. 273 - 284.

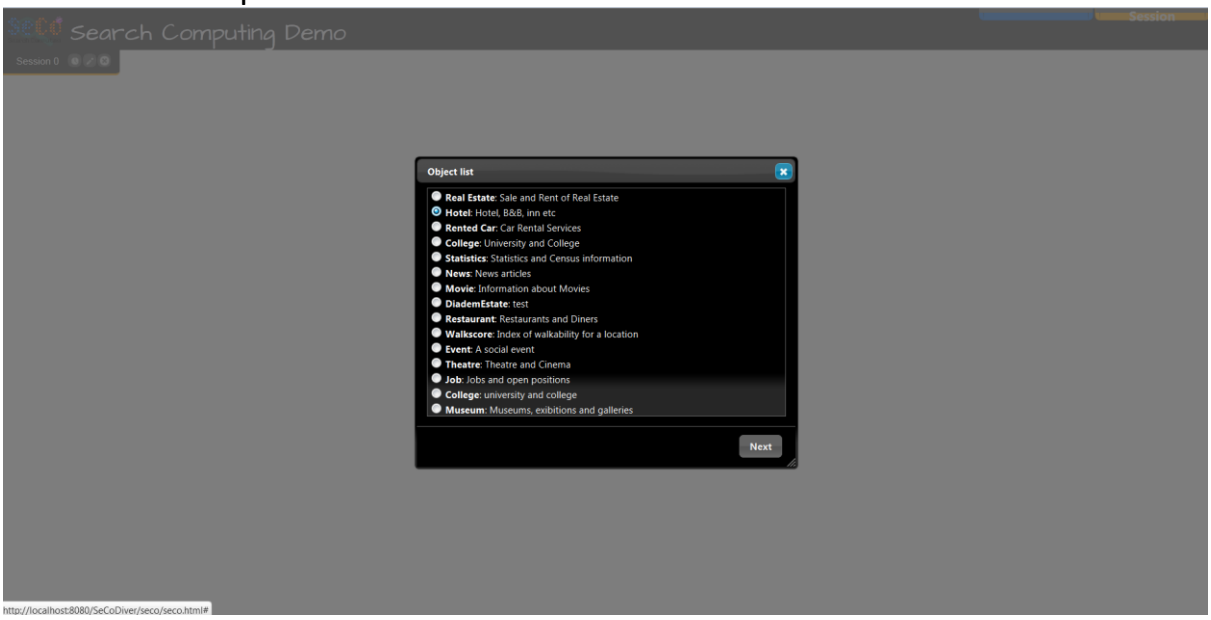
13. *Diversifying web search results.* **D. Rafiei, K. Bharat, A. Shukla.** New York, NY, USA : ACM, 2010. WWW '10: Proceedings of the 19th international conference on World wide web.

14. *Divq: diversification for keyword search over structured databases.* **E. Demidova, P. Fankhauser, X. Zhou, W. Nejdl.** New York, NY, USA : ACM, 2010.

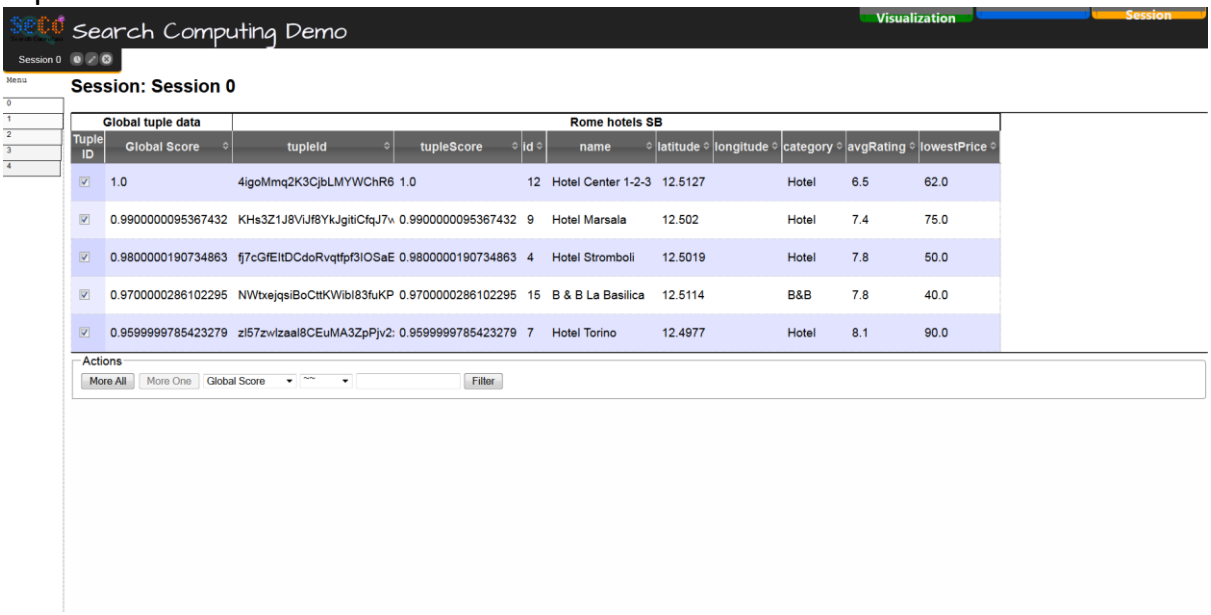
SIGIR '10: Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval. pp. 331 - 338.
15. *Structured search result differentiation.* **Z. Liu, P. Sun, Y. Chen.** s.l. : 2(1), 2009. VLDB Endow. pp. 313 - 324.
16. *Re-ranking web service search results under diverse user preferences.* **D. Skoutas, M. Alrifai, W. Nejdl.** September 2010. 4th International Workshop on Personalized Access, Profile Management, and Context Awareness in Databases (PersDB 2010).
17. *Efficient computation of diverse query results.* **E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, S. A. Yahia.** Washington, DC, USA : IEEE Computer Society, 2008. ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering. pp. 228 - 236.
18. *Using trees to depict a forest.* **B. Liu, H. V. Jagadish.** 2009. VLDB Endow. Vol. 2(1), pp. 133-144.
19. *Novelty and diversity in information retrieval evaluation.* **C. L. Clarke, M. Kolla, G. V. Cormack.** New York, NY, USA : ACM, 2008. SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval. pp. 659 - 666.
20. *Beyond independent relevance: methods and evaluation metrics for subtopic retrieval.* **C. X. Zhai, W. W. Cohen, J. Lafferty.** New York, NY, USA : ACM. SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval. pp. 10-17.
21. —. **Zhai, Cheng Xiang, William, W. Cohen and Lafferty, John.** s.l. : SIGIR '03 Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval.
22. *The use of mmr, diversity-based reranking for reordering documents and producing summaries.* **J. Carbonell, J. Goldstein.** New York, NY, USA : ACM, 1998. SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval.

Screen Shots

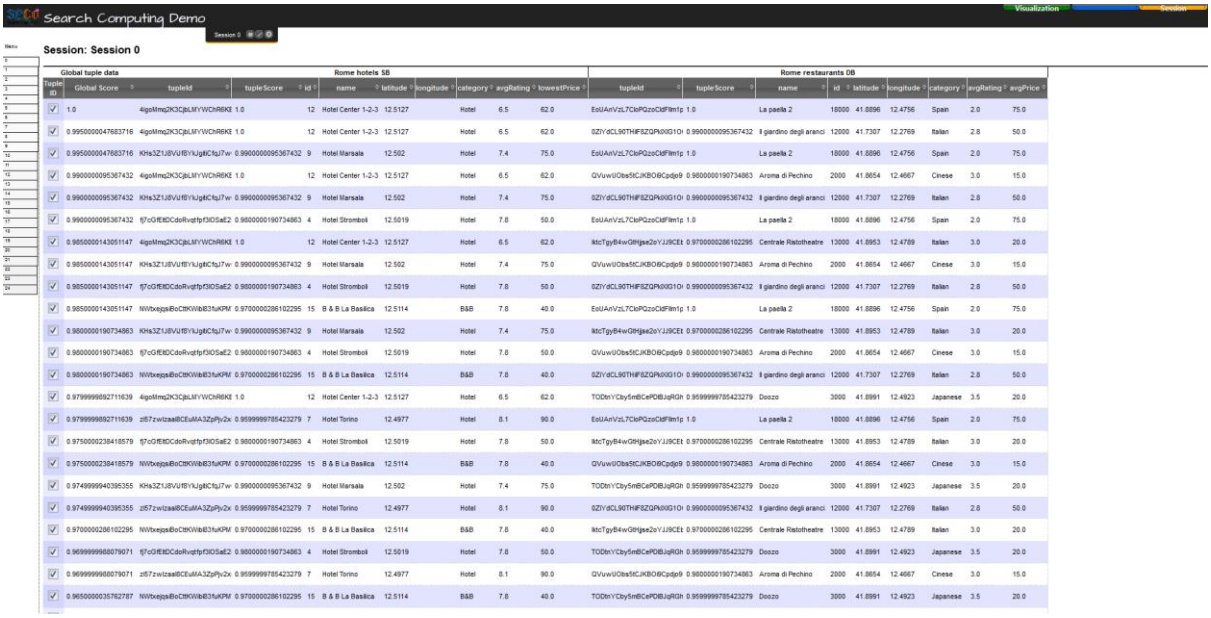
Hotel service expand



Expended hotel service

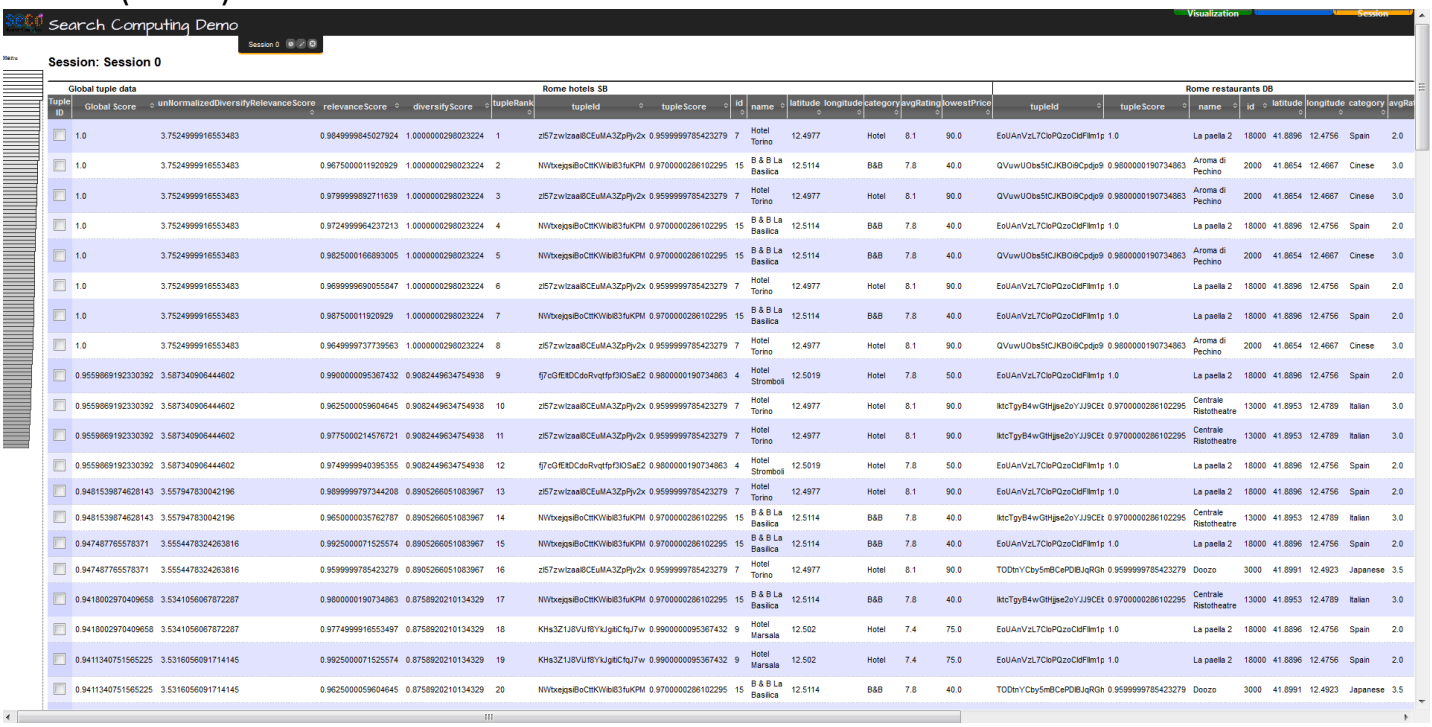


Expanded restaurant service



Tuple ID	Global Score	tupleId	tupleScore	id	name	latitude	longitude	category	avgRating	lowestPrice	tupleId	tupleScore	name	id	latitude	longitude	category	avgRating	avgPrice
☑	1.0	4igoMmq2K3CjbLMYWChREKE	1.0	12	Hotel Center 1-2-3	12.5127		Hotel	6.5	62.0	EoUAnVzL7CbPQzoCldFlm1p	1.0	La paella 2	18000	41.8896	12.4756	Spain	2.0	75.0
☑	0.9950000047683716	4igoMmq2K3CjbLMYWChREKE	1.0	12	Hotel Center 1-2-3	12.5127		Hotel	6.5	62.0	0ZIYdCL90THF8ZQPk0IG1Oi	0.9900000095367432	Il giardino degli aranci	12000	41.7307	12.2769	Italian	2.8	50.0
☑	0.9950000047683716	KHs3ZYJi8VUfBYkJgbICfqJ7w	0.9900000095367432	9	Hotel Marsala	12.502		Hotel	7.4	75.0	EoUAnVzL7CbPQzoCldFlm1p	1.0	La paella 2	18000	41.8896	12.4756	Spain	2.0	75.0
☑	0.9900000095367432	4igoMmq2K3CjbLMYWChREKE	1.0	12	Hotel Center 1-2-3	12.5127		Hotel	6.5	62.0	QVuwUObs5fCJKBOBCpdjp9	0.9800000190734863	Aroma di Pechino	2000	41.8654	12.4667	Cinese	3.0	15.0
☑	0.9900000095367432	KHs3ZYJi8VUfBYkJgbICfqJ7w	0.9900000095367432	9	Hotel Marsala	12.502		Hotel	7.4	75.0	0ZIYdCL90THF8ZQPk0IG1Oi	0.9900000095367432	Il giardino degli aranci	12000	41.7307	12.2769	Italian	2.8	50.0
☑	0.9900000095367432	fj7oGfEtDCdoRvqtfpf3lOSaE2	0.9800000190734863	4	Hotel Stromboli	12.5019		Hotel	7.8	50.0	EoUAnVzL7CbPQzoCldFlm1p	1.0	La paella 2	18000	41.8896	12.4756	Spain	2.0	75.0
☑	0.9850000143051147	4igoMmq2K3CjbLMYWChREKE	1.0	12	Hotel Center 1-2-3	12.5127		Hotel	6.5	62.0	ItcTgyB4wGtHjjse2oYJJ9CEt	0.9700000286102295	Centrale Ristotheatre	13000	41.8953	12.4789	Italian	3.0	20.0
☑	0.9850000143051147	KHs3ZYJi8VUfBYkJgbICfqJ7w	0.9900000095367432	9	Hotel Marsala	12.502		Hotel	7.4	75.0	QVuwUObs5fCJKBOBCpdjp9	0.9800000190734863	Aroma di Pechino	2000	41.8654	12.4667	Cinese	3.0	15.0
☑	0.9850000143051147	fj7oGfEtDCdoRvqtfpf3lOSaE2	0.9800000190734863	4	Hotel Stromboli	12.5019		Hotel	7.8	50.0	0ZIYdCL90THF8ZQPk0IG1Oi	0.9900000095367432	Il giardino degli aranci	12000	41.7307	12.2769	Italian	2.8	50.0
☑	0.9850000143051147	NWbxejpaBoCtKWibB3fuKPM	0.9700000286102295	15	B & B La Basilica	12.5114		B&B	7.8	40.0	EoUAnVzL7CbPQzoCldFlm1p	1.0	La paella 2	18000	41.8896	12.4756	Spain	2.0	75.0
☑	0.9800000190734863	KHs3ZYJi8VUfBYkJgbICfqJ7w	0.9900000095367432	9	Hotel Marsala	12.502		Hotel	7.4	75.0	ItcTgyB4wGtHjjse2oYJJ9CEt	0.9700000286102295	Centrale Ristotheatre	13000	41.8953	12.4789	Italian	3.0	20.0
☑	0.9800000190734863	fj7oGfEtDCdoRvqtfpf3lOSaE2	0.9800000190734863	4	Hotel Stromboli	12.5019		Hotel	7.8	50.0	QVuwUObs5fCJKBOBCpdjp9	0.9800000190734863	Aroma di Pechino	2000	41.8654	12.4667	Cinese	3.0	15.0
☑	0.9800000190734863	NWbxejpaBoCtKWibB3fuKPM	0.9700000286102295	15	B & B La Basilica	12.5114		B&B	7.8	40.0	0ZIYdCL90THF8ZQPk0IG1Oi	0.9900000095367432	Il giardino degli aranci	12000	41.7307	12.2769	Italian	2.8	50.0
☑	0.9799999892711639	4igoMmq2K3CjbLMYWChREKE	1.0	12	Hotel Center 1-2-3	12.5127		Hotel	6.5	62.0	TODbnYCby5mBCePDBJqRGh	0.9599999785423279	Doozo	3000	41.8991	12.4923	Japanese	3.5	20.0
☑	0.9799999892711639	zl57zwizaaiBCEuMA3ZpPjv2x	0.9599999785423279	7	Hotel Torino	12.4977		Hotel	8.1	90.0	EoUAnVzL7CbPQzoCldFlm1p	1.0	La paella 2	18000	41.8896	12.4756	Spain	2.0	75.0
☑	0.9750000238418579	fj7oGfEtDCdoRvqtfpf3lOSaE2	0.9800000190734863	4	Hotel Stromboli	12.5019		Hotel	7.8	50.0	ItcTgyB4wGtHjjse2oYJJ9CEt	0.9700000286102295	Centrale Ristotheatre	13000	41.8953	12.4789	Italian	3.0	20.0
☑	0.9750000238418579	NWbxejpaBoCtKWibB3fuKPM	0.9700000286102295	15	B & B La Basilica	12.5114		B&B	7.8	40.0	QVuwUObs5fCJKBOBCpdjp9	0.9800000190734863	Aroma di Pechino	2000	41.8654	12.4667	Cinese	3.0	15.0
☑	0.9749999940395355	KHs3ZYJi8VUfBYkJgbICfqJ7w	0.9900000095367432	9	Hotel Marsala	12.502		Hotel	7.4	75.0	TODbnYCby5mBCePDBJqRGh	0.9599999785423279	Doozo	3000	41.8991	12.4923	Japanese	3.5	20.0
☑	0.9749999940395355	zl57zwizaaiBCEuMA3ZpPjv2x	0.9599999785423279	7	Hotel Torino	12.4977		Hotel	8.1	90.0	0ZIYdCL90THF8ZQPk0IG1Oi	0.9900000095367432	Il giardino degli aranci	12000	41.7307	12.2769	Italian	2.8	50.0
☑	0.9700000286102295	NWbxejpaBoCtKWibB3fuKPM	0.9700000286102295	15	B & B La Basilica	12.5114		B&B	7.8	40.0	ItcTgyB4wGtHjjse2oYJJ9CEt	0.9700000286102295	Centrale Ristotheatre	13000	41.8953	12.4789	Italian	3.0	20.0
☑	0.9699999988079071	fj7oGfEtDCdoRvqtfpf3lOSaE2	0.9800000190734863	4	Hotel Stromboli	12.5019		Hotel	7.8	50.0	TODbnYCby5mBCePDBJqRGh	0.9599999785423279	Doozo	3000	41.8991	12.4923	Japanese	3.5	20.0
☑	0.9699999988079071	zl57zwizaaiBCEuMA3ZpPjv2x	0.9599999785423279	7	Hotel Torino	12.4977		Hotel	8.1	90.0	QVuwUObs5fCJKBOBCpdjp9	0.9800000190734863	Aroma di Pechino	2000	41.8654	12.4667	Cinese	3.0	15.0
☑	0.9650000035762787	NWbxejpaBoCtKWibB3fuKPM	0.9700000286102295	15	B & B La Basilica	12.5114		B&B	7.8	40.0	TODbnYCby5mBCePDBJqRGh	0.9599999785423279	Doozo	3000	41.8991	12.4923	Japanese	3.5	20.0

Expanded museum service (e.g. quantitative diversification with λ = 0.9) (Part 1)



(Part 2)