Materialization Strategies for Web Based Search Computing Applications

Srđan Zagorac

"A thesis submitted to Auckland University of Technology in fulfilment of the Doctor of Philosophy"

2014

School of Computer and Mathematical Sciences

Primary Supervisor: Assoc. Prof. Russel Pears

Acknowledgements

As Henry Ford said *"Anyone who stops learning is old, whether at twenty or eighty. Anyone who keeps learning stays young."* - Henry Ford

By the time I commenced my PhD I was what some would consider too old to be embarking on such a journey at that stage of my life, however life has a funny way of presenting opportunities at what seems to be the most inopportune of times. Then again this no longer comes as a surprise to me, unexpected opportunities present themselves along the way and that is life. As I have become older and - I hope - wiser, I realize that rarely does life follow the straight path of destiny without taking you down a winding road with some diversions along the way. A bit like taking the scenic route, you eventually get to your destination just some time later. This is exactly how I fell into yet another period of studies.

I commenced my academic career studying medicine in Croatia. This was interrupted by the war breaking out in my old country and my family moving to New Zealand. So with a future so clearly mapped out at first, my roadmap suddenly was diverted. I then went on to complete a Bachelor of Science degree in Physiology and Pharmacology at the University of Auckland. At that time my academic thirst somehow remained unquenched and I decided to combine my medical background with my old passion for Informatics and I embarked on yet another period of Postgraduate study this time in Computer Science.

Then as I was once again living a happy life, another opportunity presented itself and I embarked on a period of research work at the Knowledge Media Institute (KMi) part of The Open University where I worked on the Pharos Project with many inspiring people from around the world. It was some of these very people who encouraged me to consider culminating all of my years of study and research into a PhD. I would like to thank all those colleagues and friends I met over the years at KMi and Pharos who contributed to my research. It was during this time I met Alessandro who I would like to thank enormously for the opportunity to commence my research at Politecnico di Milano and consequently my PhD study. I am eternally grateful to Alessandro

because without his guidance not only with my PhD topic but his guidance on how to navigate my research in Italy I may never have come to this point in my life. I continued my PhD at Auckland University of Technology and I will forever be grateful for the help from Peter Sumich who gave his utmost support and believed in me and finally but most importantly my Supervisor Russel Pears who has been a constant source of encouragement and guidance without whom I would never be here today.

My family is incredibly important to me, and always will be. It is my parents who initially instilled my thirst for knowledge and who ensured my sister and I would have a bright and happy future. I want to thank my parents for their unfathomable love and unselfish support sacrificing everything for their children's future. I would like to particularly mention my mother Saša's courageousness and persistence in all her life endeavours, and my father Jure's analytical mind and patience. I can only hope one day I master each of these gifts.

I want to thank my sister Đorđija for her support and encouragement and always believing in her older brother. To my darling wife Josephine and Sydney, my biggest supporters, thank you so much for your belief in me, your patience and your unconditional love and support for which I am eternally grateful. I hope that my achievements will provide some inspiration to Sydney as he embarks upon his studies with passion and any big challenges he faces in his life.

Henry Ford was right I still feel young in my mind, if only the body would not ache so sometimes, but I would like to report my carpal tunnel intact surprisingly after all this writing.

I can only say after my own country going through war and how that caused my family's life to take a detour and consequently my own that I wholeheartedly agree with Confucius when he said *"Education breeds confidence. Confidence breeds hope. Hope breeds peace."* - Confucius

Abstract

In this thesis we provide a characterization of view materialization in the context of multi domain heterogeneous search application. Web data view materialization is presented as a solution for technical constraints and problems implied by the unknown structure of the web data sources. The web data materialization model extends the search computing (SeCo) multi-layered model, where the search services are registered in a service repository that describes the functional (e.g. invocation end-point, input and output attributes) information of data end-points.

Our first research goal is to solve the problem of finding a sequence of access patterns, which when executed produces a materialization output.

Our second research goal is the optimization of the materialization process so that the most optimal sequence in terms of materialization output efficiency and quality, executes at all times. As each access patterns can be mapped to several services each differentiated by its performance and materialized data domain characteristics, the services sequence needs to be monitored in materialization run-time by gathering and analysing predefined materialization metrics. According to materialization run-time metrics the materialization process is optimized by switching between available services.

For the first research goal we make the following novel contributions: 1) Formulation of the building blocks for the materialization feasibility analysis; 2) Definition of the materialization feasibility analysis method and the accompanying algorithms; 3) A detailed empirical study conducted on a set of materialization tasks ranging in their schema dependencies complexity.

For the second research goal we make the following novel contributions: 1) Formulation of a set of performance dimensions and their metrics for web source materialization; 2) A cost model that utilizes optimization metrics in order to qualitatively differentiate between web services in terms of materialization time; 3) A query optimization procedure that explores the characteristics of the underlying source data domain in order to prioritize the execution of the most productive queries in terms of their data harvesting power; 4) Materialization process optimization strategies based on the web

source performance dimension metrics and query optimization procedure; 5) A detailed empirical study conducted on several relevant web based data sources that clearly shows the effectiveness of the proposed solution.

Contents

List of Figures

List of Tables

Attestation of Authorship

"I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning."

Srđan Zagorac

Signed: _____

Auckland 24/11/2015

Chapter 1

Introduction

1.1 Overview

Web based data sources represent a segment of the web commonly interfaced by HTML forms or web service APIs. Recent research has estimated the size of this type of web data to be an order of magnitude larger than the size of the surface web [Madhavan 2008] i.e., the information accessible via the static URL links. This vast amount of data is commonly referred to as the deep or hidden web. In other words the deep web might be seen as a myriad of heterogeneous web based data sources commonly accessible via web forms or web API calls.

The aim of search computing [Ceri 2010] is to answer multi domain queries by accessing heterogeneous web based data sources typically found in the deep web. The usual scenario for answering such queries is to identify domains contributing to answering of the query; identify web based data sources related to those domains; query the web sources by invoking their web forms or APIs; collect the results and join them in order to answer the original query.

A major difficulty facing multi-domain query search is our inability to control query execution upon its submission to a remote data source. Often, the search process is affected by the high latency due to the Internet network congestion, routing problems or simply slow responsiveness of the 'other' side. Involved services may also restrict the number of daily accesses or impose time delays between search requests. Further, the obtained data might suffer from low coverage and high level of duplicates, thereby requiring multiple invocations of multiple sources, if available, in order to harvest the required data corpus for answering the given query.

Evidently such constraints act detrimentally to the quality of the search

process in terms of result delivery time and the result quality itself as the decreased availability of web sources might cause just a subset of the required result to be obtained.

Search Computing (SeCo) [Ceri 2010] addresses the specific problem of search service integration in order to answer multi-domain queries: the task in hand is a complex data integration process, where data comes from Web based data sources (commonly accessible through Web forms or Web APIs) that return paginated and ranked result lists. Unfortunately the access to Web data repositories is typically constrained by the data provider, e.g., by limiting the number and frequency of allowed daily service invocations; moreover, the Web service query interface constrains the set of accessible data to a subset of the whole corpus.

Obviously, the existence and quality of a search process depends on those constraints: as users expect good results provided in a timely and reliable way, a response delay or the unavailability of the allocated data sources degrades the performance of the search, thus, slowing down the processing of search results and, ultimately, causing degraded user experience. In order to alleviate the problems possibly introduced by these technical constraints while providing an efficient execution environment for multi-domain query answering, we aim at designing a solution for materializing queries as views, i.e., locally stored answers that can be, in turn, queried.

Techniques for dealing with high latency data sources involve caching mechanisms that materialize the result of queries over the data sources. While caching can increase the performance of a search system [Cambazoglu 2010], its effectiveness can be limited by its very nature; typically, cache items (query results) are stored according to a hash that is calculated from the query. While several techniques (e.g., cold start), can be employed in order for the cache infrastructure to become effective, cache items are not reused unless the queries issued are identical to those results already cached, thus, severely limiting the utility of the caching mechanism.

Data integration systems working on open Web data sources like SeCo instead, need to have the capability of exploiting already answered queries as well in order to compute a response to different queries that may be presented

to the engine in the future. Recently there have been efforts to integrate a vast amount of structured data found on the Internet in the form of HTML tables [Cafarella 2009] and by surfacing information hidden behind web forms [Madhavan 2008]. To the best of our knowledge, none of the proposed approaches addressed the materialization of data provided through search interfaces where results are ranked and paginated for the purposes of integration.

An alternative approach can be found in the literature of data integration systems where data access limitations due to schema normalization or data distribution are addressed using view materialization. A view can be materialized by storing the tuples of the view in the database or any persistent storage medium. Index structures can be built on the materialized views, thereby providing a much faster access to the materialized data. The data integration systems described in [Levy 1996, Duschka 1997a, Kwok 1996, Lambrecht 1999] followed an approach in which the contents of the data sources were described as views over a mediated schema. The mediated schema acted as a virtual bridge between the actual search application and the remote sources. Other works consider answering queries over materialized views as a key role in developing methods for semantic data caching in client-server systems [Jonsson 1996, Keller 1996, Adali 1996].

In this thesis we elaborate on the challenges raised and present solutions to the problem of query results materialization in a multi-domain search setting such as SeCo.

1.2 Motivation

Search Computing is concretely implemented through a search computing application, a vertical Web search system that leverages on the SeCo framework for enabling multi-domain search capabilities. The application typically resides on a SeCo installation and consists of a configuration of one or more multi-domain queries over the existing service repository.

As a search computing application deals with a large amount of concurrent end user requests, a sub-second response time and scalability are of primary importance. Therefore, high-performance architectures and deployment envi-

ronments able to satisfy these requirements must be part of the solution.

Short response times and time-to-screen are crucial to guarantee system responsiveness. These parameters are affected by two main factors in SeCo: internal query processing time and remote services invocation time. The former can be reduced by executing a query on multiple nodes in parallel by exploiting inter-query and also intra-query parallelism.

Service invocation time instead, can be reduced by minimizing and optimizing communications with services, possibly avoiding them altogether. At a physical level, invocation times can be reduced by efficiently using available communication protocols. HTTP in particular, provides facilities for caching Web server responses and pipelining requests to Web servers [EvenDar 2007]. At a higher level the communication problem has been addressed in meta-search systems, where a crawl meta-search hybrid approach [Craswell 2004] has been proposed to reduce Web search costs by indexing low-turnover and small data sources while meta-search the other ones. In a meta-search, the query is forwarded from the central server to each of the component sites and their returned answers are merged and presented as if they had been identified centrally.

A similar approach can be adopted for Search Computing by recurring to *materialization* of frequently accessed services that provides access to small amounts of data changing infrequently.

Hence, to efficiently execute search requests and deliver answers to the users in a timely manner, SeCo search application requires an efficient and reliable data access mechanism. In order to overcome the constraints imposed by these uncertainties, while providing an efficient execution environment for multi-domain query answering, we intend to define and characterize a system for materializing queries as views, i.e., locally stored answers that can, in turn, be queried. Such a system has to be able to source the already answered queries as seen in typical caching approaches as well as for any future queries that might be presented to the search engine.

We approach the web data source query materialization as a novel concept that has not yet been explored as a solution to the aforementioned problem.

By looking at online web caching and view materialization our intention

is to combine the most desirable features from each into a novel web data materialization approach which will satisfy search application need for unconstrained data access.

Whilst caching techniques offer advantages such as faster access to cached resources therefore saving on network resources or imposing controls on the content access and providing resources when the original web source is unavailable, they are constrained by the one to one query-result nature as the cached result is unique to this query and cannot be reused for other semantically similar queries.

As a contrast, a nature of the materialization view of providing a base table of the executed query enables materialized views to deliver a much more sophisticated rewrite algorithm. A cached result is only reused if the identical query or query fragment is executed again. Queries that benefit from query rewrite against materialized views may still roll-up data from materialized views, join back to tables or other materialized views and apply additional predicates, or in other words provide sophisticated rewrite capabilities.

Materialized view can be contained and containing, thus, providing subset or superset of the queried result. They can also be further expanded providing options for materializing a view for a particular 'expanded query', in order to answer several containing queries. For instance materialization of a query 'Select all restaurants in Auckland' against some restaurant rating web service can expand 'all' to any restaurant type, while location 'Auckland' expands to any areas within Auckland. Thus, we can provide answers to queries about all Indian or Italian or Thai restaurants in Ponsonby, CBD or any other Auckland area as the original materialized query contains a wide geographic and restaurant type area. This feature is particularly valuable as in practice one cannot afford to materialize all possible views in a real-life system. The reason is that (at least) two major types of system resources may be insufficient for servicing the selected views: (1) the storage space needed for the materialized views; and (2) the system costs of maintaining the materialized views (to keep them up to date) with respect to changing base data.

Further, a materialized view preserves the data in the database storage, whereas on-line result caches are in memory. They do not use more disk

space and they disappear when the database instance is shut down or the space inside the result cache is exhausted. Materialized view represents a base table of the executed query therefore, it is possible to build indexes against the tabled view and further apply DB self-tuning and self-management techniques such as AutoAdmin [AutoAdmin 2014] or IndexAdvisor [IndexAdvisor 2014].

From the maintenance perspective a cashed result can be replaced as a whole and refreshed at some regular time intervals, whereas materialized views can be updated on a more atomic result row by result row basis. Thus, materialized view provides means of fine tuning change propagation. The nature of the materialization view in providing a base table for the executed query also enables us to handle duplicates in materialized data set, and to better cope with computation of view derivations [Gupta 1999].

Transposition of the concept of view materialization as a natural embodiment of the ideas of pre-computation and caching in databases into the deep web domain represents the first and to our knowledge unique attempt.

Challenges. We observe web materialization challenges through a viewpoint of SeCo search application. Search applications approach the web data corpus via the multi-layered model of the SeCo Service Description Framework (SDF) [Brambilla 2011]. Here search services are typically registered in a service repository that describes the functional (e.g., invocation end-point, input and output attributes) information of data end-points. To illustrate the search process in the SDF context we consider a SeCo query expressed in a natural language "What are the cinemas showing comedies in Auckland". At the top, conceptual level of the SDF model - the query features as:
SELECT Cinemas showing comedies in Auckland. At the middle, access pattern level - the query is further differentiated by the input and output attributes of the related access patterns MovieByTitle (AP1) and CinemaByCity(AP2)
AP1 MovieByTitle: SELECT output attributes (Movie.Genre, Movie.Year, Movie.Rating) where input attribute(Movie.Title), order by Movie.Genre.
AP2 CinemaByCity: SELECT output attribute (Cinema.addr, Movie.Title) where input attribute(Theater.city).
Join operations between access patterns are performed via connection pat-

terns, logical connections established by means of input and output attributes in common domain. Thus, attribute Movie.Title logically joins access patterns TheaterByCity and MovieByTitle.
Final query presents as:
SELECT M.Movie.Genre, M.Movie.Year, M.Movie.Rating, T.Cinema.addr, T.Movie.Title FROM MovieByTitle as M, CinemaByCity as T WHERE M.Movie.Title=T.Movie.Title AND T.Cinema.City = 'Auckland' ORDER BY M.Movie.Genre.
Lastly the query is executed against the allocated service interface - bottom SDF level, which in turn maps to an external data source. In the example above we notice the effect of possible data retrieval delays or low quality of the retrieved data due to duplicates saturation on each of the involved services. In case the theater service is affected this prevents timely execution of the movie service that is dependent on the latter's results to form its own queries and retrieve results. Ultimately the whole search process is delayed and fails on account of untimely delivery of search results to the end-user.

Our intention is to approach materialization challenges at all SDF levels. At a physical level where service interfaces directly wrap service calls, materialization process deals with an issue of acquiring data in an efficient and effective way. Our intention is to apply methods similar to hidden web crawling techniques [Madhavan 2008] to query the target data source and iteratively uncover its content. Here the critical challenge is to acquire optimum data source coverage while maintaining a minimum level of communication with the target source. The level of coverage is measured by the completeness of the answers produced by querying the materialized data set in comparison with the answers obtained against the actual data-set. Hence, the adoption of appropriate query reinforcement techniques is essential for this challenge [Zerfos 2005, Wu 2006]. We intend to exploit semantic relations within source input and output query attributes in order to achieve desired level of query expansion, thus, retrieving the most out of each data source invocation. Another issue at the service interface level is the maintenance of currency and relevance [Peralta 2008] of the materialized data corpus. We need to be aware of changes in the actual sources and in a timely and efficient manner prop-

agate those changes within the materialized data so that the accuracy and correctness of answers is preserved.

At a logical level one or more service interfaces refer to a common access pattern described in terms of domain entities. Here we are presented with the task of integrating data obtained from different sources following the same access pattern (schema). The differences in ranking functions between the materialized data sets have to be considered and all the materialized data sources belonging to the same access pattern should be assigned the same rank. Another problem lies in the object identification and record matching. There is a level of semantic ambiguity present at a lexical, spatial and temporal level, thus, obscuring the differences or similarities within materialized data that have to be assessed and dealt with.

At a conceptual level collection of service marts serve as a hub for access patterns referring to the same types of entities. Here we deal with materialized data that shares the same focus while it is described by different access patterns and ranks. The aim is to reconcile the access pattern (or schema) differences and re-rank using one domain specific unifying ranking function. Thus, we remain with one semantically cohesive and schema uniform materialized data corpus referring to the same concept.

1.3 Research Goals

As illustrated above this sequence of service calls or communication between SeCo search and external data sources is subject to factors influencing connection pattern joins which in turn affects performance and accuracy of the search. These factors are beyond SeCo engine control mechanism and can be put into two main categories: the environmental constraints of the Internet and internal rules of the mapped sources. Often the search process is affected by the high latency due to the Internet network congestion, routing problems or simply slow responsiveness of the 'other' side. Mapped services may have their own rules as in limited number of daily accesses or self-imposed time delays between search requests.

In this thesis our first and overarching research goal is to solve the prob-

lem of finding a sequence of access patterns which when executed produces a materialization output. Our second research goal is the optimization of the materialization process so that the most optimal sequence in terms of materialization output efficiency and quality, executes at all times. As each access pattern can be mapped to several services each differentiated by its performance and materialized data domain characteristics, the services sequence needs to be monitored in materialization run-time by gathering and analyzing predefined materialization metrics. According to materialization run-time metrics the materialization process is optimized by switching between available services.

We define individual research objectives in the next section.

1.3.1 Materialization Formulation

Materialization formulation deals with building factors of the materialization solution. First, it considers all access patterns in SDF that contain the desired materialization output in their output domains. Second, access patterns are analysed in terms of satisfiability of input and output domains dependency. Third, feasibility analysis produces a combination of access patterns in the form of a reachability graph i.e., a combination of access patterns for which the full materialization is possible [Zagorac 2014]. Last, all of the services mapped to the selected access patterns are taken into consideration and used during the materialization process, according to a *run-time optimization procedure*. Figure 1.1 outlines the steps of the materialization solution. First, for

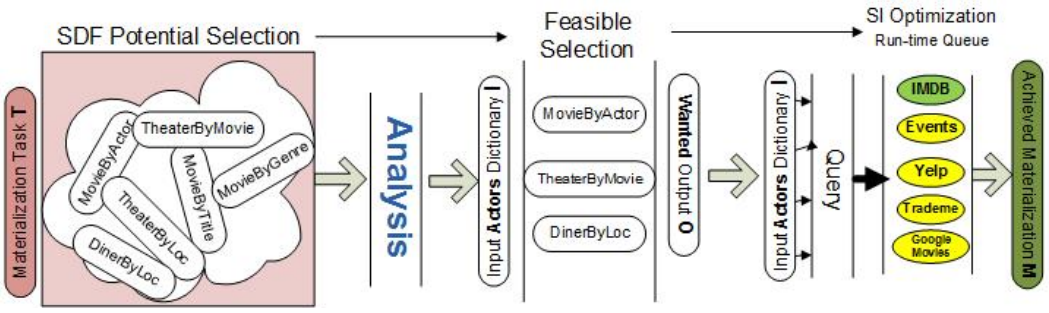


Figure 1.1: Formulation of the Materialization solution process.

a given task T it considers all access patterns in the service description framework (SDF) - described in Chapter 4 section 4.2 - that contain the wanted materialization output in their output domains. Secondly, the access pattern selection is analysed in terms of their input and output domain dependences - feasibility. Thirdly, the analysis produces the feasible combination of access patterns - a reachability graph i.e., a combination of access patterns for which the full materialization is possible. Lastly, all of the services interfaces (SI) mapped to the selected access patterns are taken into consideration and used during the materialization process, according to the run-time optimization procedure that achieves materialization M.

1.3.2 Solution Feasibility

In this thesis a feasibility analysis is performed with two distinct objectives in mind. Firstly, the objective of the analysis is to establish which of the AP combinations is capable of producing the desired materialization output for the given set of input dictionaries. In order to perform the analysis the concept of *reachability* is explored [Murata 1989]. A combination of access patterns is observed as a network in which input and output of the APs represent the nodes and the query execution instigates propagation of the values - network tokens through the network. By determining the furthest access pattern that can be reached by a single query execution the reachability of the solution is established.

Secondly, all reachable combinations of APs are further analysed to determine the network nodes whose position shapes the network coverage - *boundedness* - in terms of the number of queries this AP combination executes [Murata 1989].

1.3.3 Solution Optimization

A given materialization scenario may result in a reachability graph that involves access patterns with several available service interfaces. In this situation it is necessary to further differentiate between services belonging to the same access pattern on the basis of *service properties*.

The goal of optimization is to *minimize* the running time of the materialization process, while *maximizing* the produced materialization output and maintaining the right leverage between the performance and the quality of the obtained materialization.

As each access pattern can be mapped to several services, the materialization sequence needs to be monitored in materialization time by gathering and analysing predefined materialization metrics. According to materialization run-time metrics the materialization process can be adjusted by switching between available services so that the most optimal sequence in terms of materialization output efficiency and quality executes at all times. In this situation it is *necessary* to further differentiate between services belonging to the same access pattern.

In this thesis we look at the optimization variables and potential metrics *across service interface properties, materialized data characteristics* and properties derived from the solution feasibility analysis. Each service interface is characterized by a set of properties defining their *Uniqueness, Performance* and *Service Level Agreement* features.

Materialized data *domain characteristics* are specific to each output domain whereby each output domain logically connected to the input domain is characterized by the properties of the attribute values in the domain and how they appear in the materialization discovery. This information is used in self-sustainable materialization scenarios where output domain values are used to generate - reseed - new materialization queries.

1.4 Thesis Contributions

As a main contribution this thesis delivers a novel concept of web data view materialization as a well-defined and promising solution for ensuring the quality of the dependent heterogeneous, multi-domain query response in a search-computing application context. To our knowledge this is a first attempt in this field that provides a set of solutions unique to search computing.

The primary focus of this work is to characterize the novel concept of materializing web queries as views i.e., locally stored answers that can be queried

in order to provide a reliable data source bed for associated heterogeneous, multi-domain search.

The overall goal of the thesis is to define and characterize materialization dimensions, their properties, and their influence on effectiveness of web data materialization. This goal is driven by the motivation to overcome the constraints imposed by web data access limitations, and their effect on efficient execution of multi-domain query answering.

We make the following contributions in this thesis:

- Definition of web materialization dimensions and underlying challenges,
- Formulation, characterization and formal definition of the service and materialization properties relevant to the materialization process through all levels of SeCo Service description framework,
- Formulation of the building blocks for the materialization feasibility analysis,
- Definition of the materialization feasibility analysis method and the accompanying algorithms,
- Web data materialization optimization approach that aims at specific materialization dimensions in the materialization process context,
- Formulation and formal definition of a set of performance dimensions and their metrics for web source materialization,
- A cost model that utilizes optimization metrics in order to qualitatively differentiate between web services in terms of materialization time,
- Materialization process optimization strategies and algorithms based on the web source performance dimension metrics and query optimization procedure.

Concise materializations are required to satisfy needs of domain focused, specific results driven search scenario such as SeCo search application search. To deliver this and provide desired level of precision the search application needs

a reliable data source provider. We believe that the above achieved contributions provide the search computing process with these exact means, thus, making it valuable and complementing to the field.

1.5 Thesis Structure

This thesis consists of eight chapters, followed by appendices and references.

Chapter 1 details the research problems, questions, objectives and contributions of this thesis.

Chapter 2 gives insight into the search computing research context and the research methods that either complements or contrasts this work.

Chapter 3 emphasizes the materialization dimensions and the underlying challenges; it outlines the approaches for its resolution including the literature review of the proposed approaches.

Chapter 4 presents the building blocks of the novel concept of web data materialization. It expands on the service and materialization models; it presents service and materialization properties, materialization scenarios and the materialization process.

Chapter 5 delivers a case study that focuses on one of the building block dimensions and illustrates a potential problem solution.

Chapter 6 presents a materialization feasibly analysis; the consequent feasibility model; the empirical study and a discussion on the efficiency of the proposed model.

Chapter 7 is focused on the materialization optimization; it delivers the materialization optimization metrics, the cost model and consequent optimization algorithms. The chapter is concluded by an exhaustive empirical study proving validity of the proposed algorithms.

Finally, Chapter 8 presents a discussion of the implications of the results and a summary of the research. In addition, Chapter 8 presents a discussion of potential future research, limitations of this study and conclusions based on the contributions that have been made by this thesis.

1.6 Publications from the thesis

The following research papers have been written and published during the course of this research.

Zagorac, Srđan & Pears, Russel. **Web Data Source Materialization: Optimization Approaches.** Elsevier - Data and Knowledge Engineering Journal. *Under review.*

Zagorac, Srđan & Pears, Russel. **Web Materialization Formulation: Modelling feasible solution.** In Database and Expert Systems Applications (pp. 366-374). Springer International Publishing.

Bozzon, A., Ceri, S., & Zagorac, Srđan. **Materialization of web data sources.** In Search Computing (pp. 68-81). Springer Berlin Heidelberg, 2012

Barbieri, D. Bozzon, A. Brambilla, M. Ceri, S.; Pasini, C. Tettamanti, L. Vadacca, S. Volonterio, R. & Zagorac, Srđan. **Exploratory Multi-domain Search on Web Data Sources with Liquid Queries.** In Web Engineering (pp. 363-366). Springer Berlin Heidelberg.

1.7 Chapter Summary

This chapter has provided a summary that forms the foundation of this thesis including an introduction to the research problem and the primary research objective. The next chapter examines the web data materialization in the search computing context and presents related research areas.

Chapter 2

Methodology Overview

2.1 Introduction

In this chapter, we review main themes of literature related to the contributions made in this thesis. Firstly, we discuss previous research in the view of the materialization area that holds the most significance to this work. Then we assess significance of binding schemas (access patterns) in the materialization context.

Further, we introduce web service composition, an area of relevance to the materialization formulation feasibility solution as web data sources closely reassemble relations within web services. In the same context we introduce Petri nets as a modelling tool of preference for web services composition analysis.

Special consideration is given to query languages; we present an overview of relevant query languages and their involvement in deep web querying and data integration areas.

Further, we review previous research in the area of search engine caching in contrast to our approach. We then present the research related to the sampling of the hidden web data sources and discuss its relevance to our work. Lastly we discuss web crawling in the context of web data materialization and the data acquisition dimension.

2.2 View Materialization

Traditionally, in a relational database context views are needed because the actual schema of the DB is normalized for implementation reasons and the queries are then executed on one or more de-normalized relations that better represent the real world. Then, defining a new relation that encapsulates

the denormalized relations as a single unified view allows queries to be more intuitively specified.

A view can be materialized by storing the tuples of the view in the database or any persistent storage medium [Gupta 1999]. Index structures can be built on the materialized views [Agrawal 2000]. Consequently system accesses to the materialized view can be much faster than re-computing the view [Amiri 2003]. A materialized view is therefore like a cache - a copy of the data that can be accessed quickly.

A materialized view provides fast access to data [Aouiche 2009]; the speed difference might be critical in applications where the query rate is high or the views are complex or the physical access to data is affected by network bottlenecks or latency.

The task of view generation and materialization is mainly observed through the problem of answering queries using views. Informally the problem is as follows: Given a query Q over a data source schema and a set of view definitions $V_1, ..., V_n$ over the same schema, is it possible to answer the query using only the answers to views $V_1, ..., V_n$? Alternatively, we can also ask what is the maximal set of tuples in the answer Q that we can obtain from the views? If we have access to both the views and the data source relations, what is the cheapest query execution plan for Q?

Historically, there have been two sets of application that rely on the problem of answering queries using views. The first class, query optimization and database design, utilizes materialized views for speeding up query processing. Such savings are significant in decision support algorithms when the views and queries contain grouping and aggregation.

In the context of database design, view definitions provide a mechanism for supporting the Independence of the physical view of the data from its logical view. This independence enables us to modify the storage schema of the data without changing its logical schema and to model more complex types of indices [Valduriez 1987]. Hence several authors describe the storage schema as a set of views over the logical schema [Yang 1987, Tsatalos 1996]. Given these descriptions of the storage the problem of computing a query execution plan involves deducing how to use the views to answer the query.

The second class of applications is in the area of data integration. The task of data integration is to provide a common interface to a collection of heterogeneous data sources. Users of such systems do not pose queries in terms of a mediated schema. The mediated schema is a set of relations that is designed for a specific data integration application and contains the salient aspects of the domain under consideration. The data integration systems described in [Levy 1996, Duschka 1997a, Kwok 1996, Lambrecht 1999] follow an approach in which the contents of the sources are described as views over the mediated schema.

The data sources can vary from database systems and legacy systems to - XML based - structured files. Such sources are commonly fronted and effectively hidden by interface programs or web APIs found on World Wide Web [Baru 1999, Ives 1999, Manolescu 2001, Naughton 2001, Yu 2004].

As opposed to the traditional database application, where the database schema is modelled by the requirements of the application, the data integration encounters pre-existing data sources with its own data schemas and specifics.

2.2.1 Access Patterns

In the context of data integration where data sources are modelled as views, we may have limitations on possible access paths to data. For example when querying the Internet Movie Database (IMDB) we cannot simply ask for all the tuples in the database. Instead, we must supply one of several inputs (actor, director etc.) and obtain the set of movies in which they are involved.

We can model limited access paths by attaching a set of adornments to every data source. If a source is modelled by a view with n attributes, then an adornment consists of a string of length n, composed of the letters b (bound) and f (free). The meaning of the letter b in the adornment is that the source must be given values for the attribute in that position. The meaning of the letter f in an adornment is that the source does not have to be given a value for the attribute in that position. For example, an adornment b_f for a view (A, B) means that tuples of V can be obtained only by providing values for

attributes A.

Previous research has considered the problem of answering queries using views when the views are also associated with adornments describing limited access patterns. In [Rajaraman 1995] it is shown that the bound given in [Levy 1996] on the length of a possible rewriting does not hold anymore.

[Rajaraman 1995] shows that in the presence of access-pattern constraints it is sufficient to consider a slightly larger bound on the size of the rewriting: $n + v$, where n is the number of sub goals in the query and v is the number of variables in the query.

The situation becomes more complicated when we consider maximally contained rewritings. As the following example given in [Kwok 1996] shows, there may be no bound on the size of a rewriting. In the following example, the relation $DBpapers$ denotes the set of papers in the database field, and the relation AwardPapers stores papers that have received awards (in databases or any other field). The following views are available:

$$DBSourcef(X) : -DBpapers(X)$$
$$CitationDBbf(X, Y) : -Cites(X, Y)$$
$$AwardDBb(X) : -AwardPaper(X)$$

The first source provides all the papers in databases, and has no access-pattern limitations. The second source, when given a paper, will return all the papers that are cited by it. The third source, when given a paper, returns whether the paper is an award winner or not.
The query $Q(X)$ asks for all the papers that won awards:

$$Q(X) : -AwardPaper(X). \tag{2.1}$$

Since the view *AwardDBb(X)* requires its input to be bound, we cannot query it directly. One way to get solutions to the query is to obtain the set of all database papers from the view DBSource, and perform a dependent join with the view *AwardDBb*. Another way would be to begin by retrieving the papers in DBSource, join the result with the view CitationDB to obtain all papers cited by papers in DBSource, and then join the result with the view AwardDBb. As the rewritings below show, we can follow any length of citation

chains beginning with papers in DBSource and obtain answers to the query that were possibly not obtained by shorter chains. Hence, there is no bound on the length of a rewriting of the query using the views.

$$Q'(X) : -DBSource(X), AwardDB(X)$$
$$Q'(X) : -DBSource(V), CitationDB(V, X1), ...,$$
$$CitationDB(Xn, X), AwardDB(X).$$

However, as shown in [Duschka 1997a, Duschka 1997b], we can still find a finite rewriting of the query using the views, although a recursive one.

In [Duschka 1997a] it is shown that a maximally-contained rewriting of the query using the views can always be obtained with a recursive rewriting. In [Friedman 1997, Lambrecht 1999] the authors describe additional optimizations to this basic algorithm.

2.2.2 Query languages

Query languages can be defined as specialized programming languages for selecting and retrieving data from "information systems". These are (possibly very large) data repositories such as file systems, databases, and (are all or part of) the World Wide Web [Bailey 2005]. Query languages are specialized insofar they are simpler to use or offer only limited programming functionalities that aim at easing the selection and retrieval of data from information systems.

With the emergence of the Web in the early 1990s as a heterogeneous information source, query languages are undergoing a renaissance motivated by new objectives: Web query languages have to access structured data that are subject to structural irregularities (semi-structured data) to take into account rich textual contents while retrieving data, to deliver structured answers that may require very significant reorganizations of the data retrieved, and to perform more or less sophisticated forms of automated reasoning while accessing or delivering meta-data.

Logic-based Web query languages such as the experimental language Xcerpt [Berger 2003, Berger 2004, Guha 1998] have been proposed, and Se-

mantic Web query languages such as RQL, RDQL, and SPARQL [Prudhommeaux 2005] clearly have logical roots. Furthermore, language optimizers and evaluators of XPath and XQuery [Robie 2001, Trombetta 2004] exploit techniques formerly developed, thus, bringing these languages back to the logical roots of query languages. At the beginning of this ongoing query language renaissance, a principled and summarized presentation of query language foundations surely makes sense.

In the context of search computing, queries are expressed in a conjunctive declarative query language over service interfaces named SeCoQL [Braga 2011]. SeCoQL, an SQL variant elected as the most compact and readable conjunctive formulation for both experts and developers, easily generated by the UI modules and easily parsed by the underlying SeCo modules.

2.3 Web Service Composition

Historically, answering queries over independent data sources has been the research object of parallel or distributed query processing [Ives 2004, Özsu 2011, DeWitt 1990]. Two main techniques have emerged in this research field: code shipping and data shipping. While code shipping to Web services is not feasible, data shipping is feasible and allows the feeding of results coming from one service in the access plan to another service in the plan. The latter technique is heavily leveraged in search computing applications, as data are shipped in pipelines from one service to another, so as to maximize parallelism.

Web services are loosely coupled, platform-independent, self-describing software components that can be published, located and invoked via the Web infrastructure using a stack of standards such as SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) and UDDI (Universal Description, Discovery and Integration). Recently, Web services have been recognized as the next generation framework for building agile distributed applications over the Internet. One of the most promising advantages of Web service technology is the possibility of creating value-added services by composing existing ones. Several techniques have been proposed in this area [Dustdar 2005] [Milanovic 2004].

The coordinated execution of distributed Web services is the subject of Web services composition, which comes in two different flavours: orchestration and choreography [Daniel 2006]. The distributed approach of choreographed services (e.g., using WS-CDL [WSCDL 2005] or WSCI [WSCI 2002]) does not suit our query processing problem, because choreographies are not executable and require the awareness of and compliance with the choreography by all the involved services. The centralized approach of orchestrated services (e.g., using BPEL [OASIS 2007]) suits better the research problem addressed in this thesis, as orchestrations are executable service compositions (i.e., feasible solution in our terminology) and services need not be aware of being the object of query optimization and execution. In the specific case of BPEL, however, its workflow-based approach does not provide the necessary flexibility when the invocation order of services needs to be computed at runtime, as is our case (e.g., dynamically fixing a number of fetches to be issued to a service remains hard).

There is a growing amount of semantic approaches to the runtime composition of Web services (e.g., [WSMO 2010] or [Confalonieri 2004]), but their focus is typically on functional requirements or quality of service [Bianchini 2006] and less on data. Inspired by the work presented in [Srivastava 2006, Tatemura 2007] introduce the idea of continuous query over service-provided data feeds (e.g., through RSS or Atom). The goal is to mash up and monitor the evolution of third-party feeds and to query the obtained result. Their mash-up query model is articulated into collections of data items and collection-based streams of data (streams also track the temporal aspect of collections and allow the querying of the history of collections). Suitable select, join, map, and sort operators are provided for the two constructs. The described system consists of a visual mash-up composer, an execution engine, and interfaces for users to subscribe to mash-up feeds equipped with personalized queries.

Finally, Yahoo! Pipes [YahooPipes 2014] and IBM Damia [Altinel 2007] are a data integration services that enable a Web 2.0 approach to compose ("mash up") queries over distributed data sources like RSS/Atom feeds, comma separated values, XML files, and similar. Both approaches come with

user-friendly and intuitive Web interfaces, which allow users to draw workflow-like data feed logics based on nodes representing data sources, data transformations, operations, or calls to external Web services. Both Yahoo! Pipes and IBM Damia require the user to explicitly specify the query processing logic procedurally, which is generally not a trivial task for unskilled users, especially for the case of joins, which have to be explicitly programmed by the user. Instead, with our approach we automatically derive a plan from a declarative query formulation.

It is worth noting that the previous service querying approaches effectively enable users to distribute a query over multiple Web services, but they do not specifically focus on the peculiarities of search services, such as ranking and chunking. In this thesis we look into chunking or pagination as one of the essential service properties. Service pagination is treated as essential part of the query generation techniques and data surfacing strategies.

Also while these examples are useful first sources for looking for services, many service directories that can be found on the Web are not stable, fail to adhere to UDDI and, with time, become unreliable. Moreover, centralized registries suffer from problems associated with having centralized systems such as a single point of failure, and bottlenecks.

2.3.1 Petri Net

The Petri net [Petri 1962, Peterson 1981] is a well founded process modelling technique that has formal semantics. It has been used to model and analyse several types of processes including protocols, manufacturing systems, and business processes [Aalst 1999]. A Petri net is a directed, connected, and bipartite graph in which each node is either a place or a transition. Tokens occupy places. When there is at least one token in every place connected to a transition, we say that the transition is enabled. Any enabled transition may fire removing one token from every input place, and depositing one token in each output place.

A Web service behaviour is basically a partially ordered set of operations. Therefore, it is straightforward to map it to a Petri net. Operations are

modelled by transitions and the state of the service is modelled by places. The arrows between places and transitions are used to specify causal relationships.

Work by Benatalah and Hamadi [Hamadi 2003] assumed that a Petri net, which represents the behaviour of a service, contains one input place (i.e., a place with no incoming arcs) and one output place (i.e., a place with no outgoing arcs). A Petri net with one input place, for absorbing information, and one output place, for emitting information, will facilitate the definition of the composition operators and the analysis as well as the verification of certain properties (e.g, reachability, boundedness, deadlock, and liveness).

The work presented in this thesis relates to analysis of web data materialization feasibility as a propagation of tokens through a bipartite of interconnected places and can, thus, be modelled using Petri Nets. The main difference is the possibility of a service access pattern to contain sets of input and output places with a composition operator joining these two sets.

There are mainly two compatibility considerations [De Backer 2004, Narayanan 2002] between different web services (web data sources). First, syntactically, web services can be composed only if the provided interfaces specified by WSDL (schema) with port types operations, and message types match the required interfaces of the other web services. The second one is behavioural including equivalence and usability. Two web services are equivalent if one service can be replaced by another while the remaining components stay unchanged. Web services are usable if the interactions among them guarantee compatibility.

The syntactical compatibility refers to the conformance of access signatures between the two web services. For instance, if one web service invokes an operation of the second one, it is necessary that the parameters, their number and type match each other. Data-driven based methods are often adopted for the syntactic consideration. Data mapping is to relate equivalent data elements of two messages so that two interfaces that belong to different services can be linked. Two services can be composed by specifying the output message of one interface as the mirror image of the input message of another one. In order to solve the problems of integrating data models and message formats, Extensible Stylesheet Transformations (XSLT) can be used. Nezhad

et al. propose a solution for mismatches between service interfaces based on schema matching [Nezhad 2007]. They first identify the relationships between messages in partner web services and then specify mapping functions. A work by [Tan 2009] proposes a method to automatically check and mediate the messages sent and received by two web service parties during the interaction.

A satisfaction of syntactic requirement is not enough for successful interaction. To illustrate, let us consider two on-line restaurant rating web services with simple matching access schema that take cuisine type and location as an input and return a list of restaurants and locations ranked by user ratings. The first web service expects an area name as a location input e.g., Auckland, Ponsonby or Auckland, Parnell while the other service takes post code as a location input eg., 1010 or 1134. Consequently, interaction between these two ends up in deadlock. This kind of problem is also classified as a non-local choice problem [Ben-Abdallah 1997]. Thus, the behaviour of web services must be taken into account when analysing the compatibility of web services. Since the textual specification of BPEL is not suitable for computer aided verification for behaviour incompatibility, current methods mainly follow two steps, i.e., modelling and analysis.

As an appropriate method for modelling and analysing distributed business processes [Van der Aalst 1998], Petri nets are also an adequate modelling tool for web service behaviour. As shown in [Hamadi 2003], they are able to define and verify usability, compatibility and equivalence of web services. In particular, their semantics for BPEL are proposed [Hamadi 2003, Martens 2005b]. Since BPEL is becoming the industrial standard for modelling web service based business processes, a Petri net based method is, thus, directly applicable to real world examples.

Ouyang et al. [Ouyang 2005] transform BPEL into Petri nets represented in the Petri nets markup language and perform WofBPEL to support three types of analysis, i.e., reachability analysis, competing message-consuming activities and garbage collection of queued messages by generating the full state space. They adopt Petri net reduction rules to reduce a model before generating its state space [Ouyang 2007].

Martens [Martens 2005a, Martens 2005b] proposes a BPEL annotated

Petri net (BPN) and presents a decision algorithm for the controllability of a BPN model based on a communication graph (c-graph). A c-graph is a directed, bipartite graph in which nodes denote reachable states of the BPN and edges denote messages that the BPN is able to send or receive. Martens transforms the check of interaction between the composed BPEL processes into the verification of deadlock-freeness of a BPN. After all parts that yield a deadlock are cut off, the remaining part of BPN is proven to be controllable.

Koenig et al. [Kónig 2008] point out that the BPEL specification distinguishes two kinds of business processes, i.e., executable and abstract. Most of the previous work [Hamadi 2003, Martens 2005b] focuses on abstract ones. Their work permits the occurrence of incompatible cases. When it happens, it is detected by the analysis method based on the Petri net-based model, and then, two BPEL profiles for composition re-design can be applied, i.e., the abstract process profile for observable behaviour from bottom-up viewpoint and the abstract process one for templates from top-down viewpoint. Their main idea is to substitute an erroneous service with a correct one. The efficiency of this - substitute - based approach depends upon how to find out the exactly right web service from thousands of candidate web services with the exact behaviour that conforms to the other web services. In general, these methods may become complex, considering that a single error of a certain web service may affect the compatibility of the whole BPEL process.

We approach the web service compatibility issues by focusing on a) reachability and b) boundedness analysis in the search computing service description framework context.

2.4 Caching

Search engines use optimization in form of caching to speed up result delivery. This optimization occurs in search engines at two levels. A query enters the search engine via a query integrator node that is in charge of forwarding it to a number of machines and then combines the results returned by those machines. Before this is done, however, a lookup is performed into a cache of previously issued queries and their results. Thus, if the same query has been recently

issued, by the same or another user, then we do not have to re-compute the entire query but can simply return the cached result. This approach, called result caching, is widely used in current engines, and has also been studied by a number of researchers [Markatos 2001, Xie 2002, Lempel 2003, Baeza-Yates 2003, Fagni 2006, Baeza-Yates 2007, Baeza-Yate 2007]. A second form of caching, called index caching or list caching, is used on a lower level in each participating machine to keep the inverted lists of frequently used search terms in main memory [Jónsson 1998, Saraiva 2001, Zhang 2008].

The first published work on result caching in search engines appears to be the work of Markatos in [Markatos 2001] which studies query log distributions and compares several basic caching algorithms. The work in [Xie 2002] looks at various forms of locality in query logs and proposes to cache results closer to the user. Work by Lempel and Moran [Lempel 2003] proposes improved caching schemes for dealing with requests for additional result pages (i.e., when a user requests a second or third page of results). Several authors [Saraiva 2001, Baeza-Yates 2007, Garcia 2007] have considered the impact of combining result caching and list caching; in particular, recent work in [Baeza-Yate 2007] studies on how to best share a limited amount of memory between these two forms of caching. In [Garcia 2007], Garcia examines caches for the query evaluation process as a whole. Finally, work in [Fagni 2006, Baeza-Yates 2007] considers hybrid methods for result caching that combine a dynamic cache that exploits bursty queries with a more static cache for queries that stay popular over a longer period of time.

The work presented in this thesis bears much resemblance with the research presented in the caching area. However, the main difference being the proactive nature of the web view materialization approach that is not limited to already seen queries.

2.5 Sampling

Database Sampling is the process of randomly selecting tuples from a dataset. Traditionally, database sampling has been used to reduce the cost of retrieving data from a DBMS. Random sampling mechanisms have been stud-

ied in great detail e.g., [Chaudhuri 1995, Haas 2004, Olken 1993, Piatetsky-Shapiro 1984, Vitter 1985]. Applications of random sampling include estimation methodologies for histograms and approximate query processing using techniques [Garofalakis 2001]. However, in the case of hidden web sampling where data sources are present behind a proprietary curtain there is a need for developing a specific way of sampling underlying data with these restrictions in mind.

In web data materialization we require prior knowledge or an estimate of the remote source structure in order to execute the most expressive query in terms of retrieved result tuples and thereby enhance the materialization performance by increasing the ratio of discovered data against the number of issued queries. Alas, generating random samples from hidden databases presents significant challenges. The only view available into these databases is via the proprietary interface that allows only limited access - e.g., the owner of the database may place limits on the type of queries that can be posed, or may limit the number of tuples that can be returned, or even charge access costs, and so on. The traditional random sampling techniques that have been developed cannot be easily applied, as we do not have full access to the underlying tables.

However, these techniques do not apply to a scenario where there is an absence of direct access to the underlying database. A closely related area of sampling from a search engine index using a public interface has been addressed in [Bharat 1998] and more recently [Bar-Yossef 2008]. The technique proposed by [Adler 2001], introduces the concept of a random walk on the documents on the World Wide Web using the top-k results from a search engine. However, this document-based model is not directly applicable to hidden databases. In contrast to the database scenario, the document space is not available as a direct input in the web model. This leads to the use of estimation techniques, which work on assumptions of uniformity of common words across documents. Random sampling techniques on graphs have been implemented using Markov Chain Monte Carlo techniques, e.g., Metropolis Hastings [Metropolis 1953, Hastings 1970] techniques and Acceptance/Rejection technique [Von Neumann 1951]. Hidden databases represent

a major part of the World Wide Web and are commonly referred to as the hidden web. The size and nature of the hidden web has been addressed in [DeepWeb 2009, Ipeirotis 2001, Lawrence 1998, Lawrence 1999]. Probing and classification techniques on textual hidden models have been addressed by [Callan 1999] while techniques on crawling the hidden web were studied by Raghavan and Garcia-Molina [Raghavan 2000].

In this thesis we treat a materialization of the remote source as graph traversal of the underlying domains (domain diagram) consequently graph sampling techniques of interest as well.

Sampling on graphs has been used in many different flavours but very little has been done on matching a large set of graph properties. Previous work focused on using sampling to condense the graph to allow for better visualization [Gilbert 2004, Rafiei 2005]. Works on graph compression focused on transforming the graph to speed up algorithms [Feder 1995]. Techniques for efficiently storing and retrieving the web-graph were also studied in [Adler 2001]. Internet modelling community [Krishnamurthy 2005] studied sampling from undirected graphs and concluded that some graph properties can be preserved by random-node selection with sample sizes down to 30%.

Of most interest to this work is research presented in [Leskovec 2006] where two approaches to graph sampling are presented: under the Scale-down goal - to match the static target graph, while under the Back-in-time goal - to match its temporal evolution.

2.6 Web Crawling

A web crawler (also known as a robot or a spider) is a system for the bulk downloading of web pages. Web Crawlers are typically one of the main components of web search engines. Search engines use a crawling process to assemble a corpus of web pages, index them, and allow users to issue queries against the index and find the web pages that match the queries. A related use is web archiving (a service provided by e.g., the Internet archive [InternetArchive 2014]), where large sets of web pages are periodically collected and archived for posterity. Further use is web data mining, where web pages

are analysed for statistical properties, or where data analytics is performed on them (an example would be Atributor [Atributor 2014], a company that monitors the web for copyright and trademark infringements). Finally, web-monitoring services allow their clients to submit standing queries, or triggers, and they continuously crawl the web and notify clients of pages that match those queries (an example would be GigaAlert [GigaAlert 2014]).

Crawlers that automatically fill in HTML forms and Web APIs to reach the content behind them are called hidden web or deep web crawlers. The deep web crawling problem is closely related to the problem known as federated search or distributed information retrieval [Callan 2000], in which a mediator forwards user queries to multiple searchable collections, and combines the results before presenting them to the user. The crawling approach can be thought of as an eager alternative, in which content is collected in advance and organized in a unified index prior to retrieval. Web materialization data acquisition in many cases reassembles deep web crawling in this case applied to a Service description framework (SDF) specific set of web data sources.

Content is either unstructured (e.g., free-form text) or structured (e.g., data records with typed fields). Similarly, the form interface used to query the content is either unstructured (i.e., a single query box that accepts a free-form query string) or structured (i.e., multiple query boxes that pertain to different aspects of the content).

A news archive contains content that is primarily unstructured (of course, some structure is present, e.g., title, date, author). In conjunction with a simple textual search interface, a news archive constitutes an example of an unstructured-content/unstructured-query deep web site. A more advanced query interface might permit structured restrictions on attributes that are extractable from the unstructured content, such as language, geographical references, and media type, yielding an unstructured-content/structured-query instance. Recently, there have been efforts to integrate a vast amount of structured data found on the Internet in the form of HTML tables [Cafarella 2009] and by surfacing information hidden behind web forms [Raghavan 2000]. However, the proposed approaches do not address the materialization of data provided through search interfaces, where result data surfacing

[Cafarella 2009] is obstructed by search interfaces that impose limits on page size, the number of duplicates and service responsiveness.

A product/service review site, on-line shopping or holidays booking sites have relatively structured content (product names, numerical reviews, reviewer reputation, and prices, in addition to free-form textual comments).

The main approach to extracting structured content from a deep web site proceeds in four steps:

(1) Select a subset of form elements to populate, or perhaps multiple such subsets. This is largely an open problem, where the goals are to: (a) avoid form elements that merely affect the presentation of results (e.g., sorting by price versus popularity); and (b) avoid including correlated elements, which artificially increase the dimensionality of the search space [Madhavan 2008]. In SeCo these questions are resolved within SDF; in this thesis we further consider feasibility of interconnected form elements (connection patterns) in the materialization formulation.

(2) If possible, decipher the role of each of the targeted form elements (e.g., book author versus publication date), or at least understand their domains (proper nouns versus dates). Raghavan and Garcia-Molina [Raghavan 2000] and several subsequent papers studied this difficult problem. We illustrate the importance of this step in Chapter 5 where we present a case study related to domain discovery and data surfacing materialization properties.

Domain discovery from hidden databases belongs to the area of information extraction and deep web analytics [Chang 2006, Doan 2006, Jin 2011]. To the best of our knowledge, the only prior work that tackles one of the problems we address, namely discovering attribute domains of hidden databases is [Madhavan 2008, Shokouhi 2006].

(3) Create an initial database of valid data values (e.g., "Star Wars" and 1978 in the structured case; English words in the unstructured case). Some sources of this information include [Raghavan 2000]: (a) a human administrator; (b) non-deep web online content, e.g., a dictionary (for unstructured keywords) or someone's list of favorite authors; (c) drop-down menus for populating form elements (e.g., a drop-down list of publishers). Web data materialization touches on these issues within Data Acquisition strategies as presented

in Chapter 3.

(4) Use the database to issue queries to the deep web site (e.g., movie genre = "SF"), parse the result and extract new data values to insert into the database (e.g., movie title = "Star Wars"), and repeat. We cover data storage and query issuing as a part of materialization process described in Chapter 4. We elaborate on Step 4, which has been studied under (variations of) the following model of deep web content and queries [Zerfos 2005, Wu 2006].

A deep web site contains one or more content items, which are either unstructured documents or structured data records. A content item contains individual data values, which are text terms in the unstructured case, or data record elements like author names and dates in the structured case. Data values and content values are related via a bipartite graph as detailed in Chapter 6.

A query consists of a single data value V submitted to the form interface, which retrieves the set of content items directly connected to V via edges in the graph called V's result set. Each query incurs some cost to the crawler, typically dominated by the overhead of downloading and processing each member of the result set, and hence modelled as being linearly proportional to result cardinality.

Under this model, the deep web crawling problem can be cast as a weighted set-cover problem: Select a minimum-cost subset of data values that cover all content items. Unfortunately, unlike in the usual set-cover scenario, in our case the graph is only partially known at the outset, and must be uncovered progressively during the course of the crawl. Hence, adaptive graph traversal strategies are required.

A simple greedy traversal strategy was proposed by Barbosa and Freire [Barbosa 2004] for the unstructured case: At each step the crawler issues as a query the highest-frequency keyword that has not yet been issued, where keyword frequency is estimated by counting occurrences in documents retrieved so far. In the bipartite graph formulation, this strategy is equivalent to selecting the data value vertex of highest degree according to the set of edges uncovered so far.

A similar strategy was proposed by Wu et al. [Wu 2006] for the structured

case along with a refinement in which the crawler bypasses data values that are highly correlated with ones that have already been selected, in the sense that they connect to highly overlapping sets of content items.

Ntoulas et al. [Zerfos 2005] proposed statistical models for estimating the number of previously unseen content items that a particular data value is likely to cover, focusing on the unstructured case.

Google's deep web crawler [Madhavan 2008] uses techniques similar to the ones described above, but adapted to extract a small amount of content from a large number (millions) of sites, rather than aiming for extensive coverage of a handful of sites.

In most cases, the domain of the input attributes is not completely known, but it is possible to exploit a dictionary of keywords, or the values produced by the query results to generate legal input values [Madhavan 2008,Zerfos 2005]; in particular, a "reseed" occurs whenever the input values to be used for given calls are extracted from the results of previous calls. In general, materialization can be built by exploiting the entire set of available input data in the data source. The graph of connections linking the input data to the output often exhibits the presence of components, or "data islands" [Wu 2006]. Data islands are common to the data sources in which the underlying database graphs are not fully connected. The reseeding problem has been considered in [Calı 2009] tackling a setting where data sources can be accessed in limited ways due to the presence of input parameters. Methods used in that paper adopt recursive evaluation even in non-recursive queries, and use functional and inclusion dependencies for improving the access to data sources. Our work shares several assumptions with [Calı 2009], but also considers the additional problems such as materialization efficiency in terms of number of executed queries and the execution time brought by the invocation limitations of the queried Web services.

2.7 Chapter Summary

This chapter presented a review of view materialization concept with focus on schema mediation, query containment and binding schemas. It brought

an overview of relevant query languages and their significance in deep web querying. Further, the chapter presented a literature review in contrast to our work of the caching, sampling and web crawling research areas.

The next Chapter brings us back to the search computing paradigm, its aims, problems and areas of interest. The full focus is given to the web data materialization dimensions and our approach towards their challenges.

Chapter 3

Web Materialization in Search Computing Context

3.1 Introduction

Search Computing is a new paradigm for composing search services. While state-of-art search systems answer generic or domain-specific queries, Search Computing enables the answering of questions via a myriad of dynamically selected, cooperating search services that are correlated by means of join operations.

The latest years have witnessed an exponential increase in the number of available Web data services. Such a trend has pushed the evolution of a new classes of data integration systems, like mash-up applications or, more recently, multi-domain search applications. Unfortunately, the access to Web data repositories is typically limited 1) by the access constraints imposed by the Web service query interface, and 2) by technical limitations such as the network latency, as well as the number and frequency of allowed daily service invocations. In order to create efficient and scalable multi-domain search applications, where users expect good results provided in a timely and reliable way, there is a need to provide methods and systems that allow fast access to materialized copies of the original data.

In this chapter we provide an insight into the search computing concept; we address the problem of Web data materialization in the context of the multi-domain query systems, highlighting the underlying challenges and discussing the most relevant approaches that could be adopted to increase the performance of data materialization systems.

3.2 Search Computing

The aim of Search Computing is to respond to *multi-domain queries*, i.e., queries over multiple semantic fields of interest. Its goal is to help users by decomposing queries and to automatically assemble complete results from partial answers. Hence, 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.

Some 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 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 a minimum consists of understanding their underlying domains, in routing appropriate query subsets to each domain specific source and in combining answers from each expert domain to build a complete answer that is meaningful for the user.

The main idea behind search computing is a data integration process as complex queries are extracted from complex data and complex data demand integration. In search computing, integration is query-specific, as answering queries related to different semantic domains demands for intrinsically different data sources: building results for such queries does not require "global data integration", but simply data integration relative to specific domains.

In Search Computing a data source is any data collection accessible on the Web. The Search Computing vision is that each data source should be focused on its single domain of expertise (e.g., travel, music, shows, food, movies, health, genetic diseases, etc.), but pairs of data sources that share

information (e.g., about locations, people, genes) that 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 function alterations that are treated by drugs), then we can answer queries that connect A to C (e.g., pathologies treated by drugs) and so on. Each source is in charge of its own maintenance policy thus determining its level of data accuracy and currency.

The search computing paradigm also looks into a problem of 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. SeCo resorts to join [Ilyas 2004], which is however revisited in the context of Search Computing to become service-based and ranking-aware [Polyzotis 2011]. 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 [DBLife 2014]), inexpensive flights (extracted from a flight selection site, e.g., Expedia [Expedia 2014] or Edreams [Edreams 2014], and cities with nice beaches (extracted from tourism or review sites, e.g., Yahoo! Travel or Tripadvisor) [YahooTravel 2014, TripAdvisor 2014].

Inter domain connections contain semantics: flights connect pairs of cities at given dates; therefore connections use "dates" and "cities" as matching properties. In SeCo joins are applied in the context of web services. The assumption is that every data source is wrapped as a web service. Such services, typically, expose a query-like interface, which undertakes keyword-based input and yields ranked results as output. Services are further composed by using a ranking-preserving join. This operation is referred to as a join of

search services [Bozzon 2011a, Bozzon 2010, Braga 2011].

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 purposes, 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 [Brambilla 2011]. This vision is consistent with the emerging idea of moving from an Internet of (disorganized) pages to an Internet of (semantically coherent) objects.

Complex queries are not only hard to answer, but they can be also difficult to formulate for the user. Thus, an important research issue is to how best to capture user's search intent and direct the user towards the discovery of their 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 [Bozzon 2010, Bozzon 2011b].

Finally, Search computing considers the possibility of automatically inferring the relevant network of data sources required to build the answer from keyword-based user queries [Brambilla 2011]. This will require "understanding" query terms and associating them to resources, through tagging, matching, and clustering techniques. Thereafter, 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 [Rajaraman 2009]. One step in this direction is to extend joins 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).

Example: To familiarize with Search Computing concepts we present a

search query example in SeCo data model context. To approach web data sources SeCo Search relies on Service Description Framework (SDF) [Brambilla 2011], a multi-layered service model. Top of SDF, a conceptual level is a simple model that characterizes real world entities used to build a domain diagram (DD) encompassing different data services called Service Marts (SM). Service Marts are structurally defined by means of attributes and their relationships. Next down, a logical level describes the access to the conceptual entities in terms of data retrieval patterns (Access Patterns, AP) described by input and output attributes. Join operations between access patterns are performed by means of attributes that share the same domains, thus, forming a connection pattern. The bottom, a physical level represents the mapping of access patterns to concrete Web data source Interfaces (SI), that incorporate access endpoints with a basic non-functional properties of the service.

To quickly illustrate let us assume the user wants to specify a query about which movies of a given genre are close to its current location in New Zealand. The query translates into natural language as "What are the cinemas showing thrillers in Auckland?". We assume a service framework, as depicted in 3.1, featuring:
1) A Movie service mart that can be accessed via access pattern MovieByTitle (AP1), returning movie details based on their title, with a single corresponding service interface mapped to http://www.imdb.com (SI1).
2) A Cinema service mart that can be accessed via TheaterByCity (AP2), returning New Zealand movie theaters close to an input position, having a service interface mapped to https://www.eventcinemas.co.nz (SI2).
Join operation is preformed via a Movie.title attribute present in input and output of AP1 and AP2. Given such services, the above query results in the SeCo query language (SeCoQL) query [Braga 2011] below, which accepts as input the movie genre, the address and city of the user's location:
DEFINE QUERY Q($X:String, $U:String, $V:String) AS SELECT M., T.* FROM AP1 (Movie.Genre: $X, Movie.Year: $Z) AS M USING S1 JOIN AP2 (Theater.Addr: $U, Theater.City: $V,Theater.Phone: $W) AS T USING S2 ON M.Movie.Title=T.Movie.Title WHERE $V=Auckland AND $X=Thriller.*
At execution time, after extracting movie and theater instances, the query

produces only those pairs with a matching movie; as such, the result may be empty.

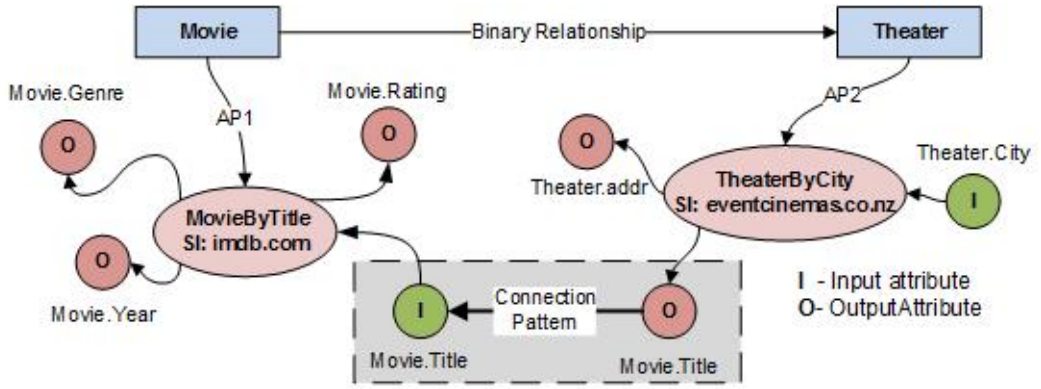


Figure 3.1: SDF illustration - MovieByTitle and TheaterByCity joined via Movie.Title connection pattern.

3.3 Web Materialization Approaches

One of the basic Search Computing requirements is a reliable, accurate and readily available data source repository that is needed to execute joins between engaged multi domain services, construct answers to the posed queries and deliver them to the end users in a timely fashion.

In order to achieve these requirements and overcome the constraints described in Chapter 1, we intend to characterize and implement a system for materializing queries as views, i.e., locally stored answers that can be, in turn, queried. Our goal is to be able to do this in an efficient way while having a good coverage of the original result set. We measure efficiency as the number of queries issued to the system that find an answer in the materialized results; coverage is measured as the degree of completeness of the answers produced by querying the materialized data set in comparison with the answers obtained against the actual dataset. We want to be able to distinguish between semantic relations found in the materialized views in order to overcome problems outlined above. We also want to be aware of the external data source updates and changes in the actual sources while efficiently propagating those changes

in a timely manner within our materialized views so that accuracy of answers is preserved.

The sources we are querying against are bound by certain input-output schema defined in the form of access patterns, where certain attributes in the data set can be used to query and rank the output (answer) and some, not necessarily the same ones, are produced in the output (answer). In order to perform materialization we, therefore, have to be aware of the effect of choosing different input attributes for querying on the output produced and how changes in the input affect the size and relevance of the output. Answers to the same query can be different depending on the ranking function and ranking attribute used from the output schema by the remote data provider source. For instance, an access pattern RestaurantByLocation&Rating that takes Location as an input and returns a results set of restaurant names, locations and customer ratings may be delivered by the remote source ranked by maximum customer rating or nearness in respect to the queried location. Clearly this aspect has to be taken into consideration as well. In this thesis we explore, define and implement novel techniques for expanding (or relaxing) the focus of the query input and output attributes in the context of answering queries using materialized views.

Ideally, the research needs to look into ways of improving expressiveness of the Search Computing Query Language (SeCoQL) [Braga 2011] used within the SeCo system for answering queries. This ought to be done by looking into ways of improving expressiveness of the language built-in types with the aim of improving their information encapsulation and thus making them semantically aware of the execution context.

3.4 Web Materialization Dimensions

The materialization, performed through data acquisition, must be executed in an efficient and effective way. Here, the critical challenge is to acquire optimum data source coverage while complying with the access constraints and minimizing the network traffic. The level of coverage can be measured as the completeness of the answers obtained from the materialized data set in relation

to what could be obtained against the actual data set. The materialization process requires the integration of the data obtained from different sources within the same AP. This task leads to a problem caused by inconsistent instances or attribute values between the materialized data sets; therefore, classic problems of object identification and record matching arise. As materialized views come in the form of tuples, duplicate detection approaches can be used to match results from multiple services. The duplicate detection is performed in the AP context and it spans different service interfaces.

To illustrate the above outlined problems let us consider a typical multi-domain SeCo query: "Where can I see a good thriller movie then have dinner in an Italian restaurant nearby, in Auckland?".

We have to query two independent web data sources: firstly, for cinemas at this location - filtered by genre - and the other one for restaurants at the same location. After we obtain both result sets we do a rank join operation on them to create an answer to the query. Clearly the join operation cannot be performed until both result sets are obtained. If we consider problems for the first category - external properties of the data source - a problem arises if the result page size of the individual result sets is unaligned i.e., the services return results of different sizes. Concretely in our example, if the theater service returns k pages with the highest score and the restaurant service returns $k/2$ pages, we then have to perform an extra Restaurant service call to get another $k/2$ result before executing the Top-k ranking join.

A similar consideration applies if there is network congestion for any of the services, the join operation has to wait until both services have returned before executing. Even worse, if one of the services has a time based call limit i.e., we can call it only a certain number of times in a given time interval (e.g., a daily limit), then if this limit is reached we have to wait until it is resolved before invoking the service again. Obviously this causes a serious time lag on the join operation and degrades performance of the search.

These factors illustrate the importance of data acquisition dimension - sourcing of the remote service data by issuing queries. We observe data acquisition through the concept of data surfacing [Madhavan 2008] which pre-computes the queries to be submitted to the web data sources of interest.

The surfacing problem consists of two main challenges: to decide which access pattern inputs to fill when submitting queries to a remote source and to find appropriate values to fill in these inputs. To provide an example let us consider RestaurantByLocation&Rating AP that takes Location and Rating as a query and returns a set of restaurant names, addresses and phones. If applied to a restaurant rating web search local to New Zealand such as yelp.co.nz [YelpNZ 2014] with query location 'London' and 5 stars rating, it is unlikely the query will produce any outcome, chances are better if we query with 'Sydney' and the best if the query location was 'Wellington'. In this case the query will reach the most in terms of result set size. The result size may further vary if we change the rating input from 5 stars to 2 or 1 star as there may not be many restaurants that bad at the queried location thus bringing even more complexity to the acquisition dimension.

Formally, we can model a service mart at the conceptual level as a database D with a single table of n attributes. At the logical level each access pattern defines a view V of D specified by different combinations of input and output attributes (or input-output schema) and shaped by a distinct ranking function. If we consider a web based data source WS that exposes a searchable interface whose input-output schema matches one of the access patterns then by executing a query Q using a particular access pattern against this data source that conforms to the binding constraints of the access pattern, we can materialize one of the views related to the service mart that this access pattern belongs to. Thus, a materialized view is specific to the access pattern, its ranking function and the data contained in the queried data source. By changing the input attributes values in the query Q we create a number of different views following the same access pattern schema and against the same web source, i.e., using the same service interface. Each of the views is unique and is referenced by the query Q and the service interface SI against which the query was executed. In turn, the service inter-face is uniquely defined by the access pattern and the web source endpoint it refers to. Hence, the combination of the query Q and the service interface descriptors is considered to be a view signature; a unique descriptor of the view. If represented in SeCoQL [Braga 2011], the view signature is the actual SI level SeCoQL query since it

holds all described signature elements. An example follows below, expressed in an SQL type language, which is enhanced with constraints on tuple size and the number of calls per time unit that can be made.

DEFINE ***QUERY*** *RestaurantPlan($X:String, $Y:String) AS SELECT R.* FROM (* ***AP2*** *(Restaurant.address: $X, Restaurant.city: $Y) AS R1 USING* ***SI2*** *)*
RANK BY (R1=0.5) LIMIT 20 TUPLES AND 50 CALLS

Theoretically, if we use all the distinct input attribute values present in a data source we end up querying the source with all possible combinations of queries, or in other words obtain all possible combinations of views of the data source for the given access pattern. Hence, we create a full materialization of this data source over the specified access pattern schema and in the context of the service mart in which the access pattern is instantiated. If extended to all queries belonging to one access pattern and provided that they are all fully rewritten we say that what is obtained is a full rewrite or full materialization of this data source in the context of the used access pattern (schema).

Formally, a set of connected domains can be modelled as a database M where each domain $d \in R_M$ where R_M is a table of n attributes at the logical level. Each Access Pattern (AP) defines a view of a table R_M detailed by the set of input and output attributes, and the applied ranking function. Finally, at the physical level, each Service Interface is modelled as a data provider for an Access Pattern that it shares with the same schema and ranking function. The objective of web view materialization is to pre-populate the defined database and thereby provide a reliable off-line information resource.

In literature, a full materialization of a query over certain schema is also known as an equivalent rewrite of a query, and is considered as one of the main notions in the problem of answering queries using views [Halevy 2001]. **Definition:** Equivalent rewritings: Let q be a query and $V = \{v_1, ..., v_m\}$ be a set of view definitions. The query q' is an equivalent rewriting of q using V if:

- q' refers only to the views in V, and
- q' is equivalent (references the same views) to q.

In our setting, given a query q over an access pattern a, we define MV as the set of materialized views of the Access Pattern a, where each view $mv \in MV$ is calculated according to a unique combination of values for the input parameters of a. Theoretically, by specifying a domain it would be possible to query such a source with all possible combinations of inputs, thus, obtaining all the existing combinations of views for the given access pattern. Hence, it would be possible to create a *full materialization* of the data sources over the specified Access Pattern schema. Of course, if such knowledge of all input values is not known a priori, the combination of views for the given access pattern cannot be obtained.

Another dimension in the problem of answering queries using views is the concept of a maximally contained rewrite [Halevy 2001].

Definition: Maximally-contained rewritings: Let Q be a query, $V = \{V_1, ..., V_m\}$ be a set of view definitions, and L be a query language. The query q' is maximally-contained rewriting of q with respect to L if:

- q' is a query in L that refers only to the views in V,
- q' is contained in q, and
- There is no rewriting $q_1 \in L$ such that $q' \sqsubseteq q_1 \sqsubseteq q$ and q_1 is not equivalent to q'.

To illustrate a maximally contained rewrite in the Service mart context described above, let us assume that we do not know all the input attributes present in the data source but instead use a dictionary of keywords or returned result values to reseed the query [Madhavan 2008, Zerfos 2005]. In some real world sources, the data in the underlying database may not be fully connected, i.e., isolated 'data islands' [Wu 2006] may exist. We define a connection between data tuples (records) as a mutually shared attribute value; a value that is present in both records. As a result, from a limited number of input attribute values, the final materialization may represent only a fraction of the target database. Such a materialization or rewrite of a query over certain schema is referred to as a maximally contained rewrite or simply the best possible rewrite of a given query.

The data acquisition needs to be performed in an efficient manner whereby the efficiency is expressed in terms of the size of the acquired data corpus against the number of executed queries. Obviously more relevant data with less executed queries is the desired outcome.

Data acquisition as a process is characterized by three main tasks that are sequentially performed in order to obtain the materialized data corpus.

Firstly, an input values discovery phase is initiated where all the service input attributes are considered and the list of values belonging to the attribute domain is created for each attribute. We group the input selection strategy into four main categories:

- Knowledge-based input strategy - query input attributes are selected from a knowledge base organized as an ontology. For example, if the input attribute required by the service is annotated with 'City' (and providing the knowledge-base conforms to the geolocation taxonomy $Street \rightarrow Suburb \rightarrow City \rightarrow Region \rightarrow Country$), then the inputs can be selected using all cities in the KB, possibly constrained by the values of other inputs (e.g., country).

- Dictionary input strategy - an existing dictionary of relevant input terms is used to populate the input attributes in the queries.

- Reseeding input strategy - method commonly used in extracting data from the deep web, as described by the literature [Madhavan 2008]. The input attributes are selected from the results obtained by the initial 'seeding' query and consequent queries are issued in recursive fashion until all result inputs are exhausted.

- Query logs input strategy - existing SeCo engine query logs are used to populate input attributes for the queries.

Secondly, a set of queries is generated using the values from the input attribute domain in the query generation phase. In the query generation phase the attribute lists are combined in pairwise fashion to obtain all the possible combinations of queries. The number of generated queries equals k^n where k is the number of input domain attributes and n the size of the

input attribute domain. This is true only if we assume that all the input values are independent (e.g., there is no correlation between the values of a city attribute and the values of the country attribute). Indeed, one may consider the existence of correlation between the values of input attributes, thus, reducing the number of generated queries.

Our query generation techniques will rely on query reseeding approaches [Madhavan 2008] as described above and will also use our proposed enhancements to algorithms proposed in the context of deep web surfacing [Zerfos 2005, Wu 2006].

Thirdly, the data is acquired by executing the queries against the external service in the data surfacing part of the data acquisition. Data Surfacing is characterized by retrieving of the resulting materialization corpus of the remote source. It consists of invoking the external service with queries generated during query generation and thereby retrieving the query results obtaining the data materialization of the service. In cases where the reseeding strategy is used this process is intertwined with query generation since queries are generated whenever new input values for the given attribute are surfaced.

To further illustrate the reseeding strategy let us again consider RestaurantByLocation&Rating access pattern that takes Location and Rating as a query and returns a set of restaurant names, ratings, addresses and phones. This reseeding strategy is applied to a restaurant rating web search local to New Zealand such as yelp.co.nz [YelpNZ 2014] taking rating as input the number of stars in a range 1 - 5 and location as a post code. The search produces results starting with the best matches, that is a matching on the queried number of stars (rating) and post code. However, the bottom of the result set typically features not exactly matching, but results that are more removed from the original query, in this case a post code in the address neighbouring the queried post code and ratings close but not equal to that specified in the query. We capitalize on this behaviour by reusing newly discovered post codes and ratings to create new queries in pairwise fashion as described above and proceed with the data acquisition process. The process continues following the same scenario by discovering new inputs and producing new queries until the query queue is exhausted or the desired data volume is obtained.

Alongside data acquisition and containment dimensions another set of dimensions of interest is expressed through semantic differences between data sources. Let us assume there is more than one service mapped in each access pattern that can be invoked to solve the example query. While solving the example query, calls to the first restaurant service returns a result set of size n, the theater service returns a result set which is $3n$ in size; thereby requiring two more restaurant service result sets of size n for the waiting rank join operation. Since we obtained two sets of results for the same query from two different data sources we cannot be certain that the same ranking function has been applied to the result sets. Thereby the rank function aggregation dimension arises, whereby data obtained by the acquisition process is affected by the ranking function applied by the service. Such an aspect is crucial when the results obtained by several queries need to be aggregated in order to define a single information corpus. We classify this materialization challenge according to the *opacity* of a ranking function:

- Transparent rank function - the rank function belonging to the remote service is known and applicable to the materialized data. In such a scenario, the materialized version of the service behaves exactly as the original one, thus, providing a faster, unconstrained replica,
- Opaque rank function - the rank function is not explicit (e.g., Google page rank); therefore it is not possible to create a unique corpus for the query response, and, consequently, the results of each query instance are materialized (and accessed) separately.

In the restaurant example even though restaurant services conform to the same access pattern, that is the same input-output schema, the shape implied by the ranking function of the returned data tuples is different thus placing it in the 'Opaque ranking function' category. Clearly, before performing the join between theaters and restaurants we have to reconcile the ranking function difference and reshape the restaurants results using some unifying rank.

We proceed to reconcile ranking differences between views obtained against the same access pattern but with different ranking attributes in order to produce, from the ranking perspective, a cohesive searchable data corpus.

Our approach is to consider all views materialized using the same access pattern with different ranking attributes. An algorithm of interest is applied in MERGE system using the PipelineResults algorithm as described by Hristidis et al. [Hristidis 2004]. We will also look into adopting and extending the LPTA algorithm based on ideas proposed in the Threshold algorithm [Fagin 2003, Guntzer 2001, Nepal 1999] as proposed by Das et al. [Das 2006]. Further methods are described in [Fagin 2003, Guntzer 2001, Nepal 1999], where algorithms are provided to combine ranked lists of attributes in order to efficiently retrieve the top results according to an aggregate function of the attributes. In the aforementioned research, a sorted list is used for each attribute in order to efficiently retrieve the top-N ranked results from a single source.

Another dimension is the issue of duplicates in the data-sources, i.e., result tuples that represent the same entity - same restaurant in our example - but are described or named differently between sources. In other words we have to be able to disambiguate between semantically related but physically different data sources. For example, both result sets contain restaurant the 'name' attribute, one which is populated by 'Bella Napoli' in the first data source but is shortened as 'B. Naples' in the second source.

There are two possible scenarios based on the result tuple identifier availability.

- The existence of an RT Identifier duplicate detection strategy - the data source natively exposes a unique result identifier.
- The absence of an RT Identifier duplicate detection strategy - there is no unique identifier (key) present in the result tuples.

We consider duplicate detection approaches used for matching records with multiple fields. These approaches fall roughly into several categories. Probabilistic matching models exist such as the Bayes decision rule for minimum error [Winkler 2002, Du Bois Jr 1969], the Bayes decision rule for minimum cost [Duda 1973] and Decision with reject region approach [Verykios 2004]). Also applicable are supervised and semi supervised learning approaches such

as CART algorithm [Breiman 1984], usage of adaptive distance function [Cohen 2002], or semi supervised probabilistic relational model as proposed by Pasula et al. [Pasula 2002]. Of further interest are approaches relying on domain knowledge or on generic distance metrics to match records. In this category of approaches methods such as active learning techniques [Cohn 1994] and usage of decision trees to learn rules exist for matching multiple records [Tejada 2002, Yancey 2005].

In distance based techniques approaches exist such as a string-matching algorithm proposed by Monge and Elkan [Monge 1996], or a simple approach of measuring the distance between individual fields and then computing a weighted distance between the records [Dey 1998]. An alternative approach, as described in [Guha 2004] is to create a distance metric that is based on ranked list merging. Finally, in the group of unsupervised learning a bootstrapping technique as described by Verykios et al. [Verykios 2000], reinforced by clustering approaches as described in [Elfeky 2002, Ravikumar 2004], may also be used.

Once surfaced materialized data requires an optimum storage environment that includes a data structure and adequate indexing that is applied to it. The materialization data structure choice is driven by the aforementioned materialization challenges such as rank opacity; consequently we recognize two distinct materialization data structures:

- Relational Table - if the ranks between all query results are reconciled (transparent rank function scenario) then the materialized data represents a coherent and uniform data structure and a schema can be defined by the access pattern schema of the service interface against which the data was obtained. Consequently the data is stored in a relational table with schema compliant with the underlying service interface.

- Index Table - in a situation where the rank reconciliation is not possible, each query result is treated as a separate entity uniquely identified by the query input parameters and the SI id (query schema definition) against which the query was executed. Each result tuple is stored in a relational table accessed via an index table holding a unique ID of the

query and the information about the position of the result tuple in the original result set.

Upon acquisition of the required materialization, a dimension of data corpus maintenance is presented. We recognize two distinct maintenance situations depending on the transparency of the update policy of the remote data source:

- Black box - updates and changes to the remote source are unknown. Maintenance policy is defined by factors such as frequency of the data access and data currency. For example, frequently accessed and older data is refreshed more frequently,

- White box - the updates and changes to the remote source are visible and are consequently propagated to the materialized data structure. For example the maintenance policy is defined by timestamp differences between the materialized data and the remote source (as, for instance, in streams).

Maintenance of the materialized data corpus presents a considerable challenge as we want to preserve the maximum possible level of currency and relevancy of the materialized data in regards to the changes in the original sources. Ideas described in the algorithm based on the concept of access frequency and the age of the materialized entry [Cambazoglu 2010] are of particular interest. We consider eager maintenance techniques such as The Eager Compensation Algorithm family of algorithms [Zhuge 1995]; and lazy maintenance approaches as described by Zhou et al. [Zhou 2007]. Of particular interest is research described by Yi et al. [Yi 2003] focusing on run-time self-maintenance in the context of top-k materialized views. Lastly, the incremental view maintenance techniques [Gupta 1995] and other concepts as outlined by Gupta and Mumick [Gupta 1993] are also of interest.

Materialization needs to be subjected to a stringent quality assessment routine. Remote service characteristics form the base of the quality assessment metrics used in the service materialization QA routine. The metrics can be used to improve the quality of the materialization by being iteratively applied to the materialization process. For example, by improving and reinforcing

data acquisition in case the tested completeness of the materialization is low or the correctness of the test queries is low as compared to the actual service query results, thereby triggering an increase of the frequency in the data maintenance policy. The metrics are as follows:

- Completeness - a measure of the data coverage as sourced by the data acquisition routine,
- Currency - a measure of the age of the materialized data in comparison to the actual data source,
- Accuracy - a measure of semantic correctness and precision of the query answers obtained from the materialized data corpus compared against the same query answers from the actual data source.

Quality of the materialized data will be assessed in terms of precision and recall comparison between the materialized data and 'real' source search results. By doing so we will utilize concepts such as Precision at n (P@N); a measure that models how a system is used and Average Precision [Voorhees 1999]. We will also consider techniques based on Geometric Average Precision or GMAP as described in [Voorhees 2005]. We will refine the quality assessment by the statistical testing in order to eliminate possibility that observed results have occurred by chance. Some of the statistical tests of interests are The Sign test, Wilcoxon Signed-Rank test and the Student's t-test as summarized and described by Hull [Hull 1993].

3.4.1 Dependencies between dimensions

Dependencies between dimensions are illustrated in the process flow chart below. The flow chart 3.2 describes the initial materialization process taking into consideration all the dimensions, from Data Acquisition Strategy, Data Model selection and Duplicate Detection. The initial materialization is monitored by the materialization maintenance process. This process performs quality assurance analysis over the materialized data corpus upon successful materialization. In the course of this thesis we will exploit the above presented

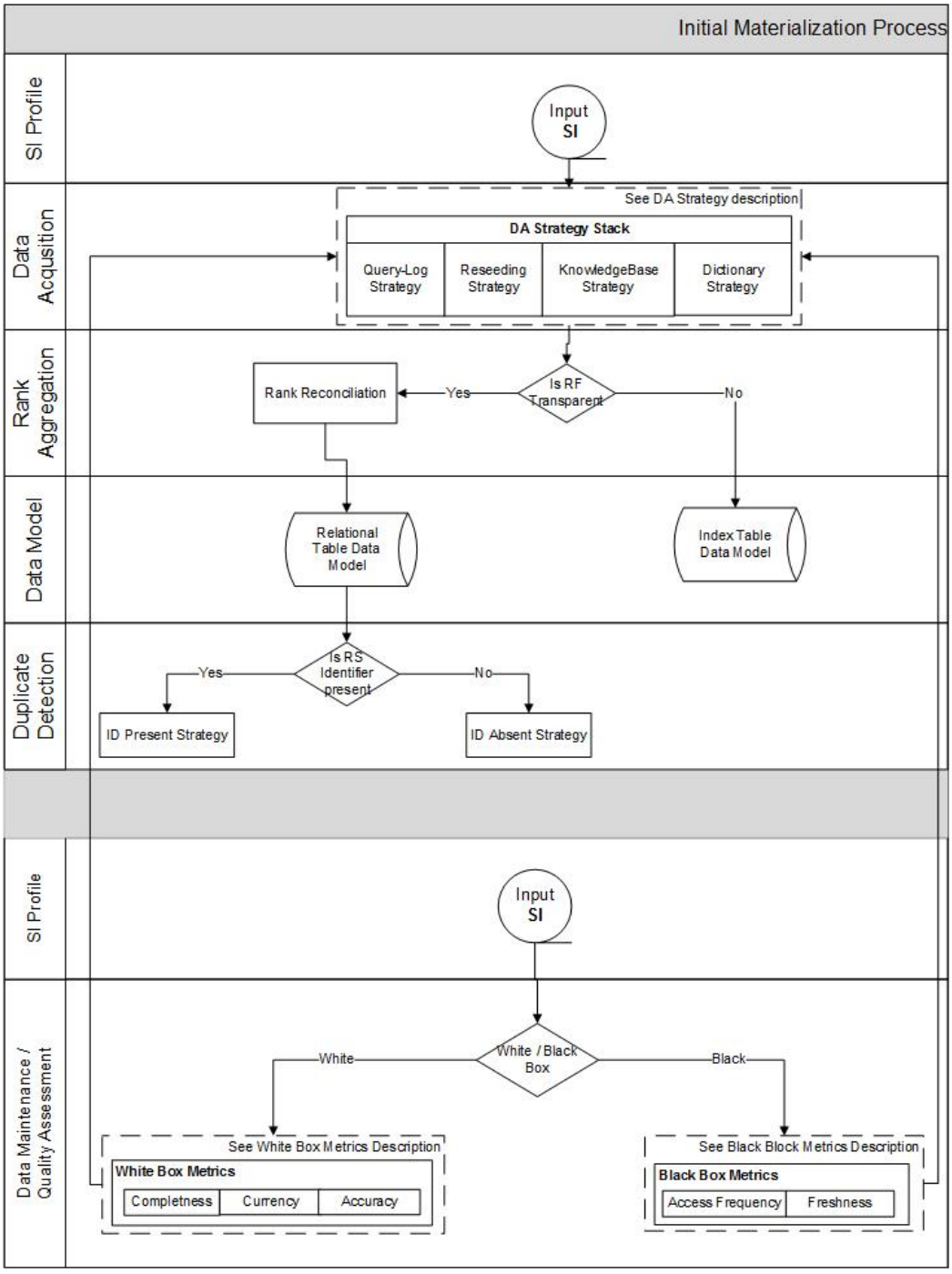


Figure 3.2: Dependencies between materialization dimensions.

concepts and approaches as required to reach the thesis goals presented in Chapter 1.

3.5 Chapter Summary

This chapter presented the search computing aims and areas of interest. It elaborated in detail on the web data materialization dimensions, their challenges and approaches in overcoming those challenges.

In the subsequent chapter we will present a full characterization of the web materialization concept with focus on service and materialization models and their properties. The next chapter will also present a case study illustrating the potential approach in solving one of the challenges of the relevant materialization dimension.

Chapter 4

Web Materialization: Building Blocks

4.1 Introduction

In this chapter we present the SeCo multi domain query framework. SeCo defines multi domain services within its own service description framework (SDF) [Brambilla 2011], which is based on a multi-layered model. Each data service available to SeCo is known in terms of the entities that it describes (conceptual view), its access pattern (logical view), the service interface, and a variety of QoS parameters (physical view).

Next we present the service materialization model (SMM). SMM is defined and implemented as a novel extension of the service description framework unique to this research. Service materialization model is dependent of SDF and operates in a seamless and complementary manner to SDF. SMM is expressed in terms of the number of used access patterns and mapped service interfaces. We closely investigate single pattern single service and multi service materialization scenarios. We also consider the single, multi and parallel queue execution models for the given scenarios. All forthcoming optimization approaches proposed in this thesis are expressed in terms of these models.

In this chapter we introduce and describe the service materialization process and how it maps to the relevant materialization dimensions introduced in Chapter 3. We observe in detail all components of the materialization process, we discuss points of performance optimization and how such process optimization might reflect on the final materialization outcome.

4.2 Service Representation in SeCo

As introduced in Chapter 3, web data materialization is a complex procedure grouped in several functionally different dimensions. Each dimension is characterized by certain properties of the web source to be materialized.

The description of such properties, required to perform materialization of Web search services, is based on the multi-layered model used in the *Service Description Framework* [Brambilla 2011] of SeCo. Search services are typically registered in a service repository that describes the functional information (e.g., invocation end-point, input and output attributes) of data end-points. At this point we will briefly describe all levels of SDF with emphasis on the features relevant to the service materialization dimensions. Figure 4.1 shows an example of *Service Description* for a movie search application. Next, we describe its features bottom-up, thereby using the same approach that is used for registering services in the SeCo system and the materialization module. At the physical level, search services are registered as Service

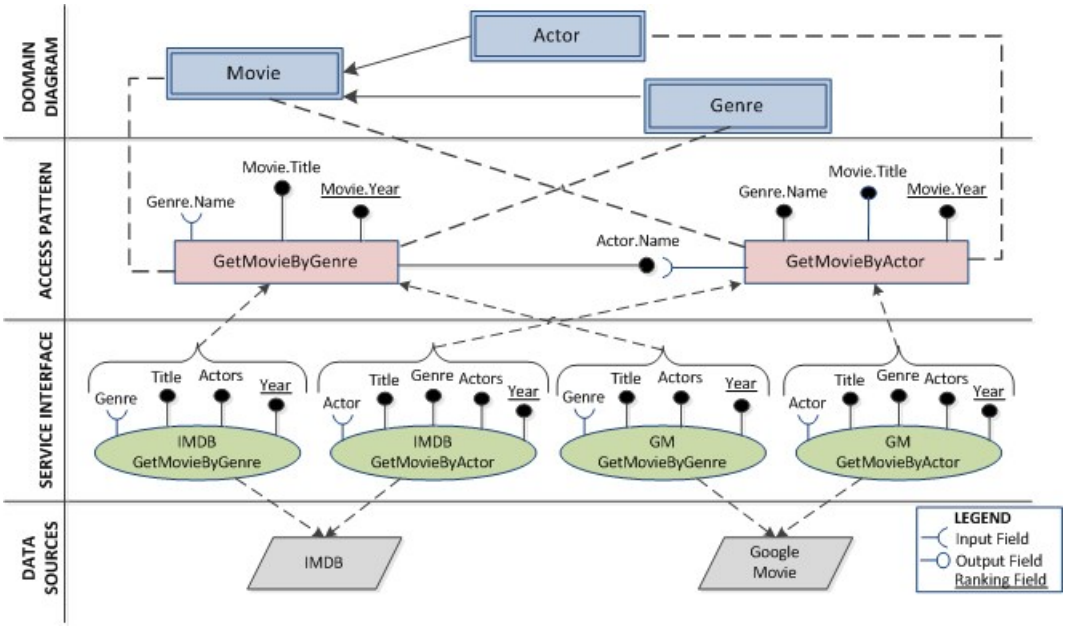


Figure 4.1: An Example of Service Description Framework for a Movie Search Application.

Interfaces (SI), a concrete representation of the service that provides a service identifier. An SI provides a set of input and output parameters and a set of

ranking attributes that specifies the ordering imposed on the service results. For instance, the SDF in Figure 4.1 contains four service interfaces working on two data sources that contain information about movies, "IMDB" and "Google Movie" respectively. Two of them query the data source and require as input the genre of the movie, while two require the name of an actor. As output, services return information about the movies that match the query condition (i.e. movies of a given genre, or movies acted in by a given actor).

At the *logical level*, services are described in terms of data retrieval patterns, or *Access Patterns* (AP). Each AP is related to one or more SI that shares the same invocation signature (input, output, and ranking attributes). Each input and output parameter of an SI is mapped to exactly one attribute of the AP.

For instance, in the case of service interface 'IMDB GetMovieByGenre' input attribute Genre maps to *Genre.Name* input attribute of the access pattern *GetMovieByGenre*; equivalently output attributes Title, Actors and Year map to *Movie.Title, Movie.Year* and *Actor.Name* outputs of *GetMovieByGenre* AP. Likewise, SI 'GM GetMovieByActor' input attribute Actor maps to input attribute *Actor.Name* of AP GetMovieByActor while output attributes Title, Genre, Actors and Year map to access pattern's output attributes *Genre.Name, Movie.Title* and *Movie.Year*.

To enrich the description of search services, APs are annotated with *entities*, properties and relationships of existing external knowledge bases (KBs) or ontologies, so as to define a *conceptual layer*, known as a *Domain Diagram*. The purpose of this annotation is two-fold: on the one hand, to provide a common ground for unifying the terminology between APs (attributes in the APs are denoted by prefixing their name with the name of the entity they refer to, e.g., *Movie.title*) and SIs; on the other hand, to support the query process by providing a richer description of the objects (and object instances) addressed by the SDF. Currently, SeCo uses YAGO ontology [Suchanek 2007] within its data model. YAGO merges Wikipedia and WordNet, while enforcing a hierarchy of *data types* to which all objects in SDF are associated. It also defines a number of *core relationships* such as *type, subclassOf, domain* and *range*. Lastly, YAGO defines relationships over relationships thus enabling

subrelationOf relationships.

Figure 4.1 shows a Domain Diagram containing three domains defined as triples $(Movie, STRING, yg\_id)$, $(Actor, STRING, yg\_id)$ and $(Genre, STRING, yg\_id)$ where (i) *Movie*, *Actor* and *Genre* are the respective domain names;
(ii) STRING denotes their types as enforced in the mapped ontology and
(iii) yg_id represents the id of the mapped ontology.

In the SeCo model, an *access pattern* (*ap*) for a relation r over a set of attributes A is an ordered pair (I, O), where I and O are disjoint sets of attributes from A, such that $I \bigcup O = A$.
Each input and output parameter of an SI is mapped to exactly one input or output attribute of the *AP*; for instance, the *GetByMovieActor* access pattern includes the IMDB *GetMovieByGenre* and the IMDB *GetMovieByActor* service interfaces. Access patterns are linked at a logical level through a *connection pattern* which is a description of the pairwise attribute relations that enable the logical connections between data sources to be established during search.

Figure 4.1 illustrates a logical connection between the 'same domain' attribute in input and output of access patterns GetMovieByGenre and GetMovieByActor, whereby the pair-wise connection is formed via attribute *Actor.Name* contained in both input and output of these access patterns. This inter-domain connectivity within the same access pattern is exploited in the reseeding input discovery during the materialization.

Each AP provides an invocation signature to one or more *Service Interfaces* (SI), a representation of the concrete data sources defined at the *physical level* of SDF. Alongside access pattern supplied invocation signature, *Service interfaces* describe profile properties of data sources, such as their actual access details, performance characteristics and their service level agreement details.

In the example above service interface IMDB encapsulates the access pattern GetMovieByGenre. The IMDB *service property* values *maxPageSize, maxResultSize, noOfChunks* and *callsPerDay* are chosen from a set of arbitrary values. These properties directly affect materialization dimensions by either acting beneficially or detrimentally to the performance of the

materialization process. The configuration illustrates one of the possible 'materialization generous' service interface profiles as SLA limit is set to 100000 calls per day and maximum page size to 100 items, which with a chunk size of 10 gives a potential maximum result size per query of 1000, thus, enabling the maximum possible surfacing of the data source. Hence, this alignment of properties acts beneficially to the data acquisition dimension as it improves the efficiency of materialization process. The wanted materialized volume is obtained with minimal number of issued queries - more obtained data with less issued queries. The IMDB invocation signature follows as:
$IMDB(http://www.imdb.com/, (100, 1000, 10, 100000), GetMovieByGenreID)$.
Comparatively, the service interface GoogleMovie maps to the same GetMovieByGenre access pattern as
$GM(http://google.com/, (10, 50, 5, 100), GetMovieByGenreID)$. The GM service interface exemplifies a restrictive service profile with very limited surfacing power as its SLA daily limit is set to 100 calls per day and a considerably smaller page and result set size of 10 and 50 respectively. In contrast, this alignment of service properties acts detrimentally to the data acquisition by decreasing the performance efficiency of the materialization as high number of queries is needed to reach the wanted materialization volume - less obtained data with more issued queries.

4.2.1 A Multi-level Model for Data Materialization

Our intention is to capitalize on the *Service Description Framework* provided by SeCo by enriching the description of search services with non-functional information (such as average response time, invocation constraints, etc.) and the description of access patterns with information related to the materialization dimensions. This information will further be applied to some of the dimensions concretely defined within the materialization process to deliver materialization efficiency optimizations in terms of execution time and data volume query ratio of the materialized web data sources.

4.2.1.1 Representing Service Properties

In the SDF, *Service Interfaces* - existing descriptors also contain information useful for the purposes of materialization. Such information can be used to drive the selection of the service interface(s) that needs to be invoked in order to efficiently materialize the addressed data source(s). Even within the SeCo search architecture these descriptors are used for compile-time optimization [Braga 2010b] and run-time adaptation [Braga 2010a]. In the data acquisition materialization dimension context we distinguish three main classes of properties of the SDF:

Uniqueness properties, i.e., properties that indicate if the service will return disjoint results for different inputs. We then distinguish two cases:

- *Unique*: two distinct queries cannot return the same item,
- *WithDuplicates*: two distinct queries may return the same item.

Performance properties, i.e., metrics that describe a service in terms of:

- *Pagination*: indicates if the service returns results in chunks or globally. Search services (e.g., the Google Movie search service) typically return results in pages (chunks), and the service consumer must perform several invocations in order to exhaust the query result set,
- *Maximum Result Size*: indicates the maximum number of items that can be returned by a single query,
- *Maximum Chunk Size*: indicates the maximum number of items that are returned in each chunk,
- *Response Time Statistics*: measures end-to-end response time (e.g., average response time for each chunk).

Service Level Agreement properties, i.e., properties that specify the level/quality of service offered by the data source service provider, including:

- *Daily Invocation Limits*, i.e., the maximum number of allowed service invocations per day (or time unit),
- *Chunk Invocation Limits*, i.e., the allowed number of consecutive chunk extractions,
- *Invocation Access Delay*, i.e., an invocation delay superimposed by the remote service,
- *Access Key*, i.e., a client identification value that is needed in order to access the remote source and enforce the SLA agreements.

4.2.1.2 Properties of Materialized Data

In this work we present novel extension to SDF service properties in form of access pattern descriptions. This extension delivers a set of properties related to materialization dimensions in access pattern context. It defines the relation between the materialized data and the data offered by the addressed sources, or to real-world data used as input for the materialization process.

Coverage Relative to the Full Materialization, this property denotes the ratio between the number of items in the materialization and the number of items in the potential, full materialization of the given data source. Coverage can be further refined into a query-specific coverage that expresses a ratio relative to the portion of data in the source that satisfy the query. E.g., an application may be interested in greater coverage of data which satisfies the query "Location = New York".

Coverage Relative to World's Entities, which is quite hard to evaluate, denotes the ratio between the number of materialized items and the number of real world items. Using multiple data sources describing the same real world entities can enhance this coverage. For instance, several services offering New York's "evening events" can be queried in order to produce a more comprehensive materialization. It requires duplicate elimination across different data sources.

Alignment This property is satisfied when the materialization contains the same data as the data source; we regard as "consistent" a time period in

which the source and materialization are aligned. Query-specific alignment expresses alignment relative to a given query. In case of full materialization, real-time alignment cannot be maintained. Materializations can be further characterized by: 1) delay, which indicates the maximum allowed time interval since the last consistent time; and 2) max-inconsistency, which indicates how many items can differ between the source and the materialization.

Redundancy A measure of the amount of duplicates in the materialization. It can be due to the presence of multiple sources, but also of a single source accessed via a service with duplicates. Duplicate removal in the first case may be harder due to the presence of value conflicts (e.g., distinct values for the same real world object).

Diversity This property measures how the items collected in the materialization represent the variety of data provided by the data sources according to some set of item attributes. For example, in the case of events in New York, it may be more interesting to include events of a different nature rather than all the jazz concerts.

Accuracy A measure of semantic correctness and precision of the query answers obtained from the materialized data corpus compared against the same query answers from the actual data source.

Ranking Preservation A measure of the ability of the system to preserve the original ranking of results obtained from the materialized data corpus, compared against the same query answer from the actual data source.

4.3 Service Materialization Model

Service Materialization Model (SMM) defines concepts novel to the SDF and web sources materialization in general. It also expands and complements some of the SDF definitions.

We build upon Service Description Framework provided by SeCo to enrich the description of service with non-functional materialization information such as average response time and service uniqueness, i.e., indication if the service will return disjoint results for different inputs. We also annotate access patterns with information related to the materialization of the underlying sources. As data acquisition of a data source is only possible through the ac-

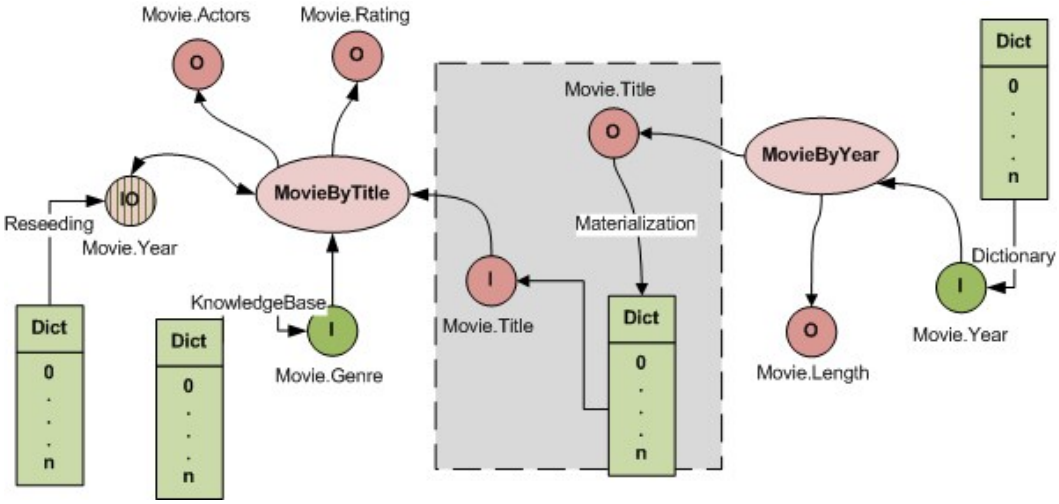


Figure 4.2: Schema representing the materialization scenario.

cess limitation imposed by Access Patterns (APs), each materialization query call requires filling the AP's input fields. To achieve coverage, the domains of all legal values for such fields must be known. Such knowledge could be available in advance when, for instance, a field insists on an enumerable value set, possibly of small size (e.g., the set of movie *genres*). Alternatively, input seeding can be seen as an *incremental process* driven by the materialization queries, where the knowledge about the input fields' domains is accumulated during this process when queries are executed and output fields are retrieved.

In the data acquisition dimension context - Figure 4.2 - we illustrate the impact of a materialization scenario on the input discovery phase by considering the two APs represented by **MovieByTitle** (which requires as input the Movie's *Title, Genre* and *Year*) and **MovieByYear** (which requires as input the movies' *Year*); the former access pattern has useful information, such as *Actors* and the *Rating*, but requires very specific input. Instead, the latter

access pattern has a simple input domain, consisting of the Year of issue (e.g., 2011); with such input, it can produce a list of titles. Movie *Genres* are few, and thus an input generator that already knows the Title and is set on the current Year can iterate over all possible genres to generate the input for the MovieByTitle access pattern, extracting all the information about actors and ratings of current Year movies.

For the benefit of future discussion, as an example of data acquisition dimension in access pattern context, we observe the access pattern: $MovieByTitle(I, O)$, where $I = Year, Genre, Title$ and $O = Actors, Year, Rating$. We note that attribute $Year$ is contained in both input and output domains, thus, enabling propagation of values via a reseed operation from the output to the input domain.
The reseeding of the output domain is reinforced by the top-k nature of the obtained results [Polyzotis 2011, Ilyas 2004]. Here just a top-k results are expected to return closest matches to the query (input), while results below this threshold will deliver different, dissimilar matches thus providing new values for the reseeding operation. This type of behaviour will be given special consideration in the remainder of the thesis.

We define the access pattern $MovieByYear(I, O)$, where $I = Year$ and $O = Title, Length$. A *connection pattern* - a logic connection - between these access patterns is established via *Movie.Title* attribute that features in both input and output domains of *MovieByTitle* and *MovieByYear* access patterns.

We also present three service interfaces mapped to *MovieByTitle* access pattern, each one featuring a different service property profile in terms of their data acquisition performance characteristics and service level agreement limits. An interface
$IMDB1(http://www.imdb.com/, (100, 1000, 10, 100000), MovieByTitle)$ features a fairly low SLA limit and a generous data acquisition performance profile of page size 100 and 10 pages for each issued query that allows for a maximum result size of 1000 tuples per query, and two restrictive interfaces
$GM1(http://google.com/, (10, 50, 5, 100), MovieByTitle)$, and
$GM2(http://google.com/, (20, 40, 2, 80), MovieByTitle)$
each featuring high daily invocation limit and rather conservative maximum

result sizes per query of 50 tuples for GM1 and 40 tuples for GM2.

In the following subsection we will outline and define SMM concepts related to data acquisition dimension that are explored within the service materialization process described further in the thesis.

4.3.1 Service Materialization Model Formalization

When considering a set of data sources and service interfaces, as in the example of Figure 4.2, several data materialization scenarios can emerge. For instance, an application might require the (full) materialization of a data source over a specific service interface (and related access pattern). In other scenarios, one might be interested in materializing the whole data source by exploiting all of its access patterns and available services (e.g., the whole IMDB database). Finally, an application might aim at collecting a comprehensive view of the information related to one or more domains (e.g., all the movies ever released).

SeCo description framework service characteristics and its novel extension of the materialization dimensions contained in access patterns are applied in the materialization process and formally defined.

Definition 4.1. In the context of a Service Materialization Model, a materialization over a given service interface s mapped to an access pattern AP is referred to as a *single pattern single service materialization* (**spss**). SPSS materialization is an ordered pair $spss\_m(s, R)$ where s is a service interface definition and $R = \{r_1 \bigcup r_2 \bigcup ... \bigcup r_n\}$ is the union of all result sets r_n obtained during materialization m.

Definition 4.2. The materialization coverage of R relative to some AP output V is defined as: $Cov = (|R|)/(|V|)$. It denotes the ratio between R - the number of tuples discovered in materialization m - and the total number of tuples in output V.

For instance, in terms of access pattern *MovieByTitle*, materialization over one of its service interfaces (e.g, IMDB1), is considered as a *single pattern single service materialization* (spss) $imdb1\_m(IMDB1, R_{IMDB1})$. The coverage $Cov_{imdb1\_m} = |R_{imdb1\_m})|/|V_{O,MovieByTitle}|$ refers to the ratio between

the numbers of tuples discovered in materialization of IMDB1 to total IMDB1 size.

Definition 4.3. A materialization query over a given access pattern AP is defined as a tuple $q^p = (v_1, ..., v_n)$, where q is a query identifier, p is the number of result pages returned by the query with $1 \leq p \leq MaxNoPages$, where $MaxNoPages$ is the maximum number of pages retrievable by a query as prescribed by the remote source; v_n is a value of attribute n in the input domain I; n is determined by the cardinality of I.

Definition 4.4. A materialization result set over AP is defined as a set of tuples $\{r(v_1, ..., v_n)\}$ where v_n is a value of attribute n in the output domain O; n is determined by the cardinality of O.

Definition 4.5. A materialization call over (si mapped to) AP is defined as an ordered pair $c(q^p, r_q)$, where (i) c is the unique call id, (ii) q^p is a query that initiated the call; and (iii) r_q is the result set - a result page p that answers q^p.

During data acquisition of an access pattern, a data source is accessed by issuing a sequence of materialization calls. Each call c contains a query q^p over this access pattern's input interface and corresponding result set r_q which is expressed over AP output interface.
Following the above we define a sequence of materialization calls as a set $\{Cm\}_{j=1}^{k}$, an ordered pair $(\{Qm\}_{j=1}^{k}, \{Rm\}_{j=1}^{k})$, where $\{Qm\}_{j=1}^{k}$ is a set of materialization queries; and $\{Rm\}_{j=1}^{k}$ is a set of materialization result sets; where k is the number of calls performed to obtain materialization m.
Furthermore, a materialization call over (si mapped to) access pattern AP is expressed in SQL-like syntax:

SELECT ***distinct*** *(<$v_{O,1}$,...,$v_{O,m}$> AS r_tmp, Rm) AS r_q FROM si_{AP} WHERE ($a_{i,1} = v_{i,1}$ &,...& $a_{i,n} = v_{i,n}$ & pageNumber = p) AS q^p*
Where (i) r_q is a result set aggregated by a duplicate tuples detection routine (distinct);
(ii) $distinct(x, y)$ aggregates result set r_q by filtering the retrieved set of

tuples $r_tmp\{< v_{O,1}, ..., v_{O,m} >\}$ from tuples already in materialization Rm;
(iii) each domain value v_m in r_tmp element belongs to attribute element of output interface O; and $m = |O|$;
(iv) q_p is expanded so that each domain value v_I is associated to the attribute element of input interface I; and $n = |I|$;
(v) p is a result page number the query refers to, $1 \leq p \leq MaxNoPages$, where *MaxNoPages* is the maximum number of pages retrievable by the query, it is defined by the service provider.

Observed through the *MovieByTitle* materialization, an example query posed against IMDB1 service may present as
$imdb1\_q001^1 <' Avatar',' action', 2009 >$, where $imdb1\_q001$ is this query's unique id, superscript 1 signifies the result page number this query relates to; and terms *Avatar, action* and *2009* are the values in the respective domains of $I_{MovieByTitle}$.
The query in SQL-like syntax features as:
SELECT ***distinct****({<Title, Genre, Year>} as r_tmp,* R_{IMDB1}*) AS* $r^1_{imdb1\_q001}$ *FROM IMDB1 WHERE (Title =' Avatar' AND Genre =' action' AND Year = 2009 AND pageNumber = 1) AS* $imdb1\_q001^1$
This query results in:
r_tmp{<'Avatar', action,2009>, <'Avatar2', action, 2016>, ...,
<'The Avatars', action, 2013>, <'Avatar Spirits', documentary, 2010>},
with the assumption that Rm already contains tuple
<'Avatar Spirits', documentary, 2010>, $distinct(r\_tmp)$ delivers the final result:
$r^1_{imdb1\_q001}$<'Avatar', action,2009>, <'Avatar2', action, 2016>,...,<'The Avatars', action, 2013>}.

Definition 4.6. A materialization is considered feasible if there is an input values dictionary (dict) allocated to all attributes in the input domain, that is $\forall a \in I \exists dict \subseteq V_a : 1 \leq |dict| \leq |V_a|$.

A feasible solution allows for materialization call set to be executed sequentially.
A materialization call set $\{Cm\}_{j=1}^{k}$ is limited by k, where k is determined by: a) the size of Cartesian product of all provided dictionaries; and b) the

number of returned result set pages, i.e., the length of the query set sequence is:
$\{Qm\}_{j=1}^{k} = dict_1 \times dict_2 \times ... \times dict_n$,
where $k = |dict_1 \times dict_2 \times ... \times dict_n| \times noOfReturnedPages$,
$n = |I|$, where $noOfReturnedPages$ is the total number of retrieved results r in materialization.
This dictates the length of $\{Rm\}_{j=1}^{k}$ which implies cardinality
$|Rm| = \sum_{i=1}^{k} pageSize_i$.
In the case of single pattern single service materialization (spss):
$V_{I,si} \subseteq V_{I,ap} and V_{O,si} \subseteq V_{O,ap}$.

Similarly, due to limiting factors such as output domain O_{si} duplicates saturation, which occurs when a data source provides result tuples that are already present in the materialization, Cov_m is decreased, thus, rendering the materialization process execution inefficient in terms of the number of executed queries and obtained materialization volume.

Let us consider a feasible materialization of $IMDB1$ with input dictionaries a subset of IMDB1 input domain $dict_{Title} = Avatar, Titanic$, $dict_{Genre} = Drama, Action, Comedy$ and $dict_{Year} = 2005, 2006, 2007, 2008, 2009$. Let us also assume for the sake of the example that $noOfReturnedPages$ is predefined as 5 pages per query. A Cartesian product of the provided values gives a query set Q_{IMDB1} of size $k = (2 \times 3 \times 5) \times 5 = 150$, and with assumption that each materialization call achieved (a) maximum result page size and (b) duplicate free result size of 100 tuples, the produced materialization R_{IMDB1} cardinality (size) was 150x100 = 15000 tuples. As the used input dictionaries contained a subset of the input domain $V_{I,IMDB1}$ the produced materialization was a fraction of the full output domain of $IMDB1$ - $V_{O,IMDB1}$, or in terms of coverage $Cov_{IMDB1} = 15000/|V_{O,IMDB1}|$. However, the materialization execution still performed to its full potential as it produced the maximum result size for the given number of executed queries. Following this example it became evident that failure of the queried data source to deliver (a) full size result pages and (b) duplicate free results sets - would act detrimentally to the materialization coverage as it would decrease the size of the final materialization.

4.4 The Materialization Process

The materialization process is a novel concept unique to this research. It deals with data acquisition and duplicate detection materialization dimensions. Within this process we use aforementioned SDF service characteristics and access pattern SMM properties to first establish the materialization process and second to deliver optimization of the materialization execution time and performance efficiency in terms of the number of queries - materialization volume obtained with minimum number of queries.

We define a data materialization process for multi-domain queries as a sequence of three data acquisition related tasks: 1) *input discovery*, 2) *query generation*, and 3) *data surfacing*. In a typical materialization process, the three tasks are performed cyclically, as depicted in Figure 4.3

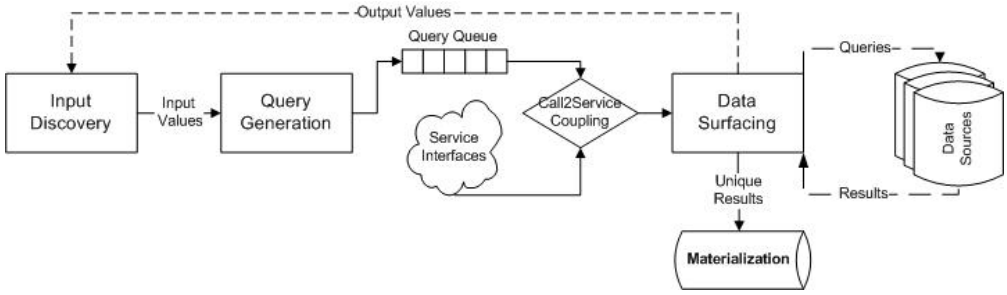


Figure 4.3: Materialization Process before optimization.

The input values discovery phase provides input values for the given access pattern input interface such $\forall a \in I \exists dict \subseteq V_a : 1 \leq |dict| \leq |V_a|$. We identify three input discovery strategies:

1. The dictionary input strategy, where a static, existing dictionary of relevant input terms is used to supply values for the input attributes of the queries;

2. The reseeding input strategy, where the input is selected from the results of previous queries. The reseeding strategy assumes a logical connection

between at least one attribute in the input and output domains. The values are discovered during data surfacing phase, domain-matched and then passed back to the appropriate attribute dictionary. The concept of reseeding within the materialization process is further illustrated later in this chapter;

3. The query logs input strategy, where existing query logs are used to populate the input attributes of the queries.

A materialization process can adopt one or more of the previously defined strategies, so as to drive the creation of the queries used to materialize the targeted data sources. For instance, a materialization process can start with a *dictionary* input strategy and then be fed with the new values provided by the *reseeding* strategy. The **query generation** phase is committed to the generation of the input values combinations used to populate the *materialization queries set {Q}* which contains a list of queries to be executed. Finally, the **data surfacing** phase is in charge of selecting which *materialization call* will be processed next. Once the call is selected the contained query is addressed against one of available service interfaces, thus, retrieving the results and producing *Rm*. Data Surfacing phase relies on the **call to service coupling** that orchestrates interaction between materialization calls and service interfaces being materialized. Call to service coupling consists of two mutually complementing parts. Firstly, a **query queue de-queuing strategy** which is in charge of query selection. In SeCo the de-queuing strategy is based on classical *breadth-first* and *depth-first* tree traversal algorithms as described in [Bozzon 2012]. Secondly, a **service interface selection strategy** is used which is responsible for the selection of the next service to be queried from the pool of available service interfaces. When the reseeding input strategy is used for one of the input attributes, data surfacing is intertwined with the input discovery phase.

As an example we consider a feasible materialization of IMDB1 with *initial* input value which is a subset of the input domain as:

$dict_{Title} = \{Avatar, Titanic\}, dict_{Genre} = \{Drama, Action, Comedy\}$ and $dict_{Year} = \{2008, 2009\}$.

In terms of the *input discovery* the process is started as a dictionary input

strategy and then continued as a reseeding Input strategy. The materialization is started by dictionary input strategy as the values are loaded from predefined initial dictionaries for each input domain. A *query generation* phase generates the initial query queue by obtaining a Cartesian product of the given initial values. A *data surfacing* phase cycle is performed by selecting one of the queries from the query queue and executing it against one of the available service interfaces - call to service coupling. The result is fetched and checked for duplicates by the **distinct** routine and then stored - duplicate detection materialization dimension. Next, the process switches back to input discovery; now as a reseeding input strategy. Since input and output domains of *MoveByTitle* share an attribute Year, all newly obtained Year values are checked against the existing $dict_{Year}$, and if not present, are added to the dictionary. As stated in Section 4.2 reseeding relies on the top-k nature of the queried service as only top-k results are set to match the query while result tuples below this threshold are likely to deliver new values in the shared domain.

Another round of query generation gets the Cartesian product with fresh Year values, assembles queries and adds them to the query queue. This is again followed by data surfacing call2service coupling and retrieving of the results, new input values (Year) discovery and query assembly. The process cycles until either the wanted materialization coverage is reached or the query queue is exhausted.

We further develop on the SMM model to provide the means of optimizing the materialization process. The SDF model is extended by expanding a domain value to an ordered pair (v, sv) where v is a constant, i.e., a value in the domain and $v \in dV$ where dV is a set of all domain values; sv is a tuple containing domain value properties.

To fully appreciate the domain value properties and their significance in the materialization process, we transform **query queue de-queuing** of the Data surfacing phase to a **tree traversal activity** based on the distribution of values in the output domain V_O.

Let us consider the following example, whereby attribute values from each materialization call form a clique [Wu 2006]. If two calls share the same attribute value, the corresponding vertex is said to "bridge" the two cliques. For

example, each of the following two calls of the materialization of MovieByTitle form a clique:

c_1 (q_1 ('Avatar', 'action', 2005),

r_1{('Avatar2', 'action', 2006),..., ('Avatar: The Last Airbender', 'animation', 2008)})

and

c_2 (q_2 ('Star Wars', 'action', 2010),

r_2{('Star Wars', video game', 1983),..., ('Star Wars: Episode III', 'action', 2008)})

Output domain value Year '2008' bridges these two cliques, that is links these two materialization calls. Our intention is to exploit this behaviour to increase the input value domain discovery rate by prioritizing queries containing such values.

Figure 4.4 shows $IMDB1$ service interface materialization with the input

q_{imdb}	r_{imdb}
q_1(2005)	r_1{(2006), (2008), (2008)}
q_2(2010)	r_2{(2008), (2009), (2011)}
q_3(2011)	r_3{(2008), (2009), (2013)}

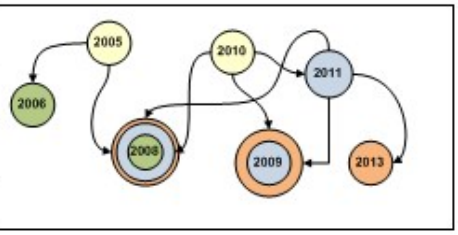


Figure 4.4: Example of the sequence of materialization calls and corresponding AVG graph.

and output connected via a $Year$ domain. The output domain is shaped by the data service provider imposed temporal radius limit of 3 years i.e., each materialization call obtains a result with a 3 year offset to the queried year. Materialization is initiated with $dict_{Year}$ initial dictionary (2005 and 2010) - presented in yellow. In the first call year 2005 is used as a query value, the returned result tuples discovered years: 2006 and 2008 - green coloured; in the second call year 2010 is used as a query value and the obtained result tuples discovered years: 2008, 2009 and 2011 - blue coloured. The third call is then executed by using reseed value 2011 from $Year$ output domain, and the fetched results contain years: 2008, 2009 and 2013 - orange coloured. The example illustrates emerging of an Attribute-Value graph (AVG) with nodes of different degrees. The vertices with years 2008 and 2009 are discovered (and linked as depicted by colouration) by more than one materialization call. The

difference of the domain value degrees in the AVG, their discovery frequency and a number of calls containing the value (harvest rate) provide the basic intuition for the domain value properties and related metrics.

Definition 4.7. A domain value property is an ordered pair defined as $sv(frequency, harvestRate)$ where *frequency* and *harvestRate* are domain value statistics defined further in the section.

Definition 4.8. A domain value *frequency f* is a property of output domain value v defined as a number of occurrences n of value v in Rm or $f = n(v)$, where $v \in V_O$. For instance materialization IMDB1 may produce $|Rm| =$ 15000. The frequency of output domain Year value 2008 is 200 if the value is present in 200 out of 15000 tuples.

Definition 4.9. A domain value *harvestRate hr* is a property of domain value v directly derived from its AVG degree. It is defined as a count of materialization calls whose result set contains value v or $hr = count(v, \{Cm\}_{j=1}^{k})$. For instance, if materialization IMDB1 executed 150 materialization calls and 30 of these calls contained Year 2008 in its output, then the harvest rate of this value is 30.

Formally, each access pattern attribute is an ordered pair $a(d, V)$ where d denotes the domain and V is a set of values. The AVG is an undirected graph that can be constructed so that each edge (E) in AVG stands for a materialization call link between vertices (V) which correspond to values in set V such as v_i and v_j. In terms of output domain attribute Year, every tuple in IMDB1 materialization call is linked with another result tuple by their respective year values.

By characterizing structured Web sources as AVGs the materialization process can be transformed into a graph traversal activity. Moreover, an access pattern materialization process becomes a self-driven graph traversal or graph crawl if the following condition is satisfied: $\forall a_i(d_i, V_i) \in I; a_j(d_j, V_j) \in O$ such that $d_i = d_j$. Thus, in essence the graph traversal is enabled by the input and output attribute being drawn from the same data domain.

Therefore, providing there is an initial dictionary of values such that ($initDict \subseteq dict$) as a starting set of seed vertices, at each step, a graph

traversal process that selects a previously seen vertex (value - $v \in V_j$) can discover all the neighbours of v. As $d_i = d_j$, the discovered vertices $v \in V_j$ are used as input dictionary dict of a_i, thus, ensuring usage of the values present in the domains of the materialized source. As described in the previous subsection, this strategy for input data surfacing is referred to as a Reseeding Input strategy. It enables materialization-time based input data discovery, and allows for dynamic expansion of the materialization call set since the calls are generated and added to the set as new values are discovered.

Depending on the web source structure, an AVG is not necessarily fully connected. It may consist of several disconnected graphs that form isolated cliques or data islands in the total data corpus [Wu 2006]. Thus an attribute value graph AVG of an attribute in the output domain of service interface may form a disjoint union of graphs U. This implies $|initDict| = |U|$ and a prior knowledge of input attributes domain distribution. Hence, either a domain sampling technique is employed or randomly chosen *initial dictionary* of appropriately large cardinality to cover for most of the possible data cliques.

To illustrate this situation let us expand the above example so that output domain Year contains a value 1997 as well as 2005, 2006, 2007, 2009, and 2011. Following the materialization with the same initDict (2005, 2009) and with the same connectedness, the produced materialization does not contain results including year 1997. Evidently, this configuration of the output domain as specified by internal data provider processing that returns results in a 3 years offset to the queried year does not link the year 1997, thereby rendering the reseeding strategy incapable of reaching this value. Hence the results containing 1997 are unreachable for the given initDict, thus, giving rise to an isolated clique and making full materialization unobtainable.

Definition 4.10. We further expand on the SDF model by defining a service interface materialization properties related to materialization m as a triple $smp(responseTimePerQuery, noUniqueTuples, pageSizePerQuery)$.

Service interface materialization properties are a set of dynamic parameters that characterize services in terms of their performance during the materialization process.

A property *pageSizePerQuery* characterizes each service interface by actual result page size $|r_{tmp}|$ prior to duplicate pruning for each materialization call. A property *pageSizePerQuery* is defined as $ps \in PS$, where (i) PS is the total number of discovered tuples in materialization and (ii) $0 \leq ps \leq maxPageSize$.

A property *noUniqueTuples* characterizes each service interface by the number of unique tuples discovered in a materialization call. A property *noUniqueTuples* is defined as $wd \in Rm$, where (i) Rm is the total number of discovered unique tuples in materialization and (ii) $0 \leq wd \leq maxPageSize$. A materialization property *responseTimePerQuery* characterizes service interface by actual per query response time. A property *responseTimePerQuery* is defined as a pair (q, rpq), where q is the id of the query and rpq its the response time, $rpq \in RPQ$, where RPQ is the total response time used by the query set sequence.

In case of the $IMDB1(http://www.imdb.com/, (100, 1000, 10, 100000), MovieByTitle)$ materialization, upon execution of a materialization call the obtained materialization properties may feature as *pageSizePerQuery=95* - the obtained result size was 95 tuples; (maxPageSize=100) and *noUniqueTuples =67* - as after duplicate pruning the remaining unique tuple count was 67. The execution time was captured from the moment the query was issued to the point when the result was registered by the process, giving a *responseTimePerQuery = 967ms*. These properties vary between calls and provide the base for qualitative service differentiation in the materialization process optimization approach.

Further, we intend to monitor the materialization process in terms of the achieved coverage over the number of executed materialization calls in a period of time. This is required to establish a basis for the qualitative analysis of the materialization and the applied optimization strategies.

Definition 4.11. A materialization efficiency Em is expressed two-fold:
(i) in terms of achieved coverage Cov_m over a number of executed materialization calls $|C_m|$ as $E_C = Cov_m/|C_m|$, and
(ii) in terms of achieved coverage Cov_m over a period of time T_m as $E_T =$

Cov_m/T_m where $T_m = \sum_{i=1}^{k} ResponseTimePerQuery_i$, and $k = |C_m|$.

4.5 Materialization Scenarios

When considering a set of data sources and service interfaces, as in the example of Figure 4.2, several data materialization scenarios can emerge. For instance, an application might require the (full) materialization of a data source over a specific service interface (and related access pattern). In other scenarios, one might be interested in materializing the whole data source by exploiting all its available services (e.g., the whole IMDB database). Finally, an application might aim at collecting a comprehensive view of the information related to one or more domains (e.g., all the movies ever released). According to the requirements of the underlying application, we identify four classes of data materialization scenarios:

Single Pattern, Single Service (SPSS): the goal is to optimize the cold-start materialization of a single service with given access pattern. In this scenario the main sources of complexity are provided by chunking of the query answer and by the distribution of values for the input attributes, which call for methods that are capable of balancing between the need for dataset coverage and dataset diversification (w.r.t. the allowed query inputs).

Single Pattern, Multiple Service (SPMS): the goal is to optimize the cold-start materialization of several services with the same access pattern insisting on different data sources. In this scenario, w.r.t. the previous one, an additional source of complexity are the record matching and duplicate identification, which are required in order to obtain a consistent cache.

Multiple Patterns, Single Service (MPSS): the goal is to optimize the cold-start of sources described by a schema with interacting APs as described in Figure 4.2, where the output values of an AP can feed the input values of another AP; each AP is mapped to a single service. The

problem may reduce to a **single data source (MPSS-1)** with multiple APs, or generalize to **multiple data sources (MPSS-M)**. Figure 4.2 represents an instance of the MPSS-1 problem if we assume in that the two APs are on the same underlying data service.

Multiple Patterns, Multiple Services (MPMS): the goal is to optimize the cold-start of sources described by a schema with interacting APs as described in Figure 4.2, where the output values of an AP can feed the input values of another AP, and each AP can be mapped to multiple services. The problem may reduce to a **single data source (MPMS-1)** with multiple APs, or generalize to **multiple data sources (MPMS-M)**.

The materialization scenarios are presented as novel contributions to the concept of web data materialization. The feasibility analysis and the optimization approaches presented in consequent chapters are observed through these scenarios.

4.5.1 Single pattern multi service (SPMS)

Single pattern multi service **SPMS** scenario depicts the situation when materialization is performed over several sources (services) mapped to same access pattern.

Definition 4.12. In SPMS scenario materialization is defined as an ordered pair $spms_m(SI, R_{spms_m})$ where:
(i) $spms_m \in SPMS\_M$ where SPMS_M is a set of all $spms$ materializations;
(ii)$1 < |SI|$, $SI \leftarrow ap$ such that $ap \in AP$ where AP is a set of all access patterns in SDF.

A materialization queries set Q_{spms_m} over many si mapped to ap is defined as:
$Q_{spms\_m} = \{Q_{si,1} \bigcup Q_{si,2} \bigcup ... \bigcup Q_{si,n}\}$ where $si \in SI$ mapped to ap and $n = |SI|$.
A materialization result set R_{spms_m} over many si mapped to ap is defined as:
$R_{spms\_m} = \{R_{si,1} \bigcup R_{si,2} \bigcup ... \bigcup R_{si,n}\}$ where $si \in SI$ mapped to ap and $n =$

$|SI|$.
Consequently, a materialization call set C_{spms_m} is defined as $C_{spms\_m} = \{C_{si,1} \bigcup C_{si,2} \bigcup ... \bigcup C_{si,n}\}$ where $si \in SI$ mapped to ap and $n = |SI|$.
Hence, a materialization call set $\{C_{spms\_m}\}_{j=1}^{k} = \{Q_{spms\_m}\}_{j=1}^{k}, \{R_{spms\_m}\}_{j=1}^{k}$ contains calls to be issued to all participating services.

4.5.2 Multi pattern multi service (MPMS)

A multi pattern multi service **MPMS** scenario depicts the situation when materialization is performed over several sources (services) mapped to several access patterns - Figure 4.2.

Definition 4.13. In MPMS scenario materialization is defined as an ordered pair $mpms\_m < SPMS\_M, CP >$ where:
(i) $SPMS\_M$ is a set of $spms\_m$ materialization where $|SPMS\_M| \leq |AP| : AP \in SDF$;
(ii) CP is a set of connection patterns where:
$\forall a_1, a_0 \in cpANDcp \in CP \exists spms\_m_1, spms\_m_0 \in SPMS\_M$.

A materialization queries set Q_{mpms_m} in MPMS scenario is defined as:
$Q_{mpms\_m} = \{Q_{spms,1} \bigcup Q_{spms,2} \bigcup ... \bigcup Q_{spms,n}\}$
where $spms\_m \in SPMS\_M, n = |SPMS\_M|$.
A materialization result set R_{mpms_m} in MPMS scenario is defined as:
$R_{mpms\_m} = \{R_{spms,1} \bigcup R_{spms,2} \bigcup ... \bigcup R_{spms,n}\}$
where $si \in SI$ mapped to ap and $n = |SI|$.
A materialization calls set C_{mpms_m} in MPMS scenario is defined as:
$C_{mpms\_m} = \{C_{spms,1} \bigcup C_{spms,2} \bigcup ... \bigcup C_{spms,n}\}$
where $si \in SI$ mapped to ap and $n = |SI|$.

4.5.3 Execution models

In the materialization process context we explore three distinct query queue execution models. The data surfacing phase of the materialization process is characterized by call2service coupling that in turn may be executed as a

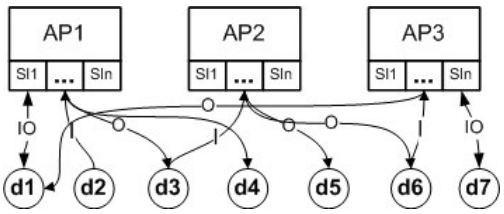


Figure 4.5: Typical multi pattern multi service (MPMS) layout with input and output domain dependencies.

serial single queue de-queuing where all participating services point to the same query set. As opposed to the serial multi queue de-queuing where each participating service has its own associated query set and conforms to the query queue de-queuing and service interface selection strategies contained within call2coupling routine. Both serial execution models are still constrained within the same data surfacing phase relying to one call2service coupling as depicted in Figures 4.6 and 4.7.

In case of a serial single queue execution the query queue de-queuing strategy optimization is performed on the basis of output domain characteristics of all services and contained within the same queue, thus, providing a 'one fits all' optimization. We expect an enhanced effect of the individual service domain properties to the query queue de-queuing strategy optimization in case of multi queue execution as each service points to its own queue featuring queue optimization specific to this service output domain. This intuition will be further explored and proved in the relevant chapter(s).

Alongside these two call2service coupling controlled de-queuing models comes the parallel execution model in which each service features its own query queue but without any particular de-queuing strategy applied, that is, queries are removed from the queue in a natural order. In parallel model each service and a query queue are executed in parallel, in its own data surfacing phase, thus, effectively performing several materialization processes in parallel.

The size of materialization call set sequence and the rate of its processing depend on materialization execution model. In the remainder of the section we further exemplify and describe in more detail all three execution models.

In **serial single queue execution model** (SSQ) there is one material-

ization call set sequence allocated to the materialization process, input value dictionary size is $|dict| = |I|$, that is a number of input dictionaries matches the number of attributes in the input interface of *ap*. As in the *spss* scenario the size of $\{C\}_{j=1}^{k}$ is limited by k, where k is determined by the size of Cartesian product of all provided dictionaries and number of discovered result pages. The size of the input dictionaries is driven by the input value discovery strategy of each $a \in I$. There are two possible outcomes: a) input values discovery strategy is *static* i.e., dictionary size is known a priori and so is the call set size, and b) the input values discovery is *dynamic* as in case of reseeding input strategy and the materialization call set size is unknown until materialization ends.

During materialization call set sequential execution, each call is executed against a service interface selected according to the supplied (optimized) call to service allocation algorithm. The call to service allocation algorithm is based on materialization properties of executing services. The shape of the call to service algorithms, their heuristics and implementation are described further in this work. To illustrate the single pattern many services scenario with

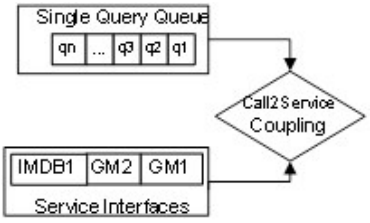


Figure 4.6: Serial Single Queue execution example.

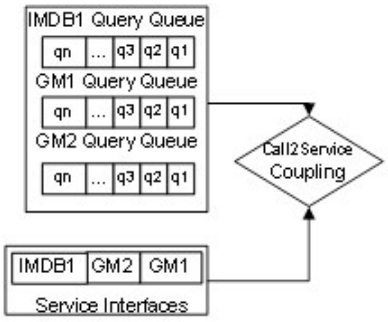


Figure 4.7: Serial Multi Queue execution example.

SSQ model we consider IMDB1, GM1 and GM2 service interfaces mapped to MovieByTitle access pattern. As in the materialization process example the materialization is started by populating a **single** query queue from initial dictionaries, as in Figure 4.2. Data surfacing query2call coupling is performed by fetching of a query from the *single query queue* that is executed against one of the participating services. Reseeding input discovery identifies a new

Year values in the output, that are further passed to the query generator for new queries to be generated and placed in the query queue. The obtained duplicate free result is stored in the provided DB storage.

In **serial multi queue execution model** (SMQ) there is a materialization call set assigned to each participating service interface. The input value dictionary size is $|dict| = |I| \times |SI|$, that is, input dictionaries are allocated to each $a \in I_s i$.

In SMQ model materialization call sequence is defined as set $(\{C_{si,n}\}_{j=1}^{k})$ where:

(i)$\{C_{si,n}\}_{j=1}^{k} \in (\{C_{SI}\}_{j=1}^{K})$;

(ii) $k = |dict_1 \times dict_2 \times ... \times dict_m|$ and $m = |I|$;

(iii)$\sum_{i=1}^{n} k_i$; (iv) $n = |SI|$.

As in SSQ model each $\{C\}_(j = 1)^k$ size is driven by the final size of $\{Q\}_(j = 1)^k$. The size of each query set is defined by Cartesian product of input value dictionary sizes for each $a \in I_s i$ and number of discovered result pages. As in SSQ model, dictionary sizes are either known a priori in case of static, predefined dictionary or unknown in case of dynamic dictionary values allocation by reseeding input discovery strategy.

As in SSQ a call to service allocation algorithm selects the call to be executed against the service. However, as opposed to SSQ, where selected services are executed against the calls from the same query set, in SMQ the choice of the service interface implies the choice of the query set as well. As in SSQ the allocation algorithm is based on materialization properties of executing services.

Serial multi queue execution model is illustrated by materialization MovieByTitle of three service interfaces IMDB1, GM1 and GM2. The process is started by creating three query queues each associated to one of the participating services. The queues are initially identical as they use the same provided initial input dictionaries, Figure 5. During the query2call coupling GM1 is invoked by executing a query from the GM1 query queue. Furthermore the values discovered by reseeding are used to populate GM1 query queue. The obtained duplicate free result is stored in the provided DB storage. In the next materialization loop IMDB1 service might be used in the

call2service coupling, this time IMDB1 query queue is replenished by new queries obtained by reseeding input discovery.

In **parallel queue execution model** (PQ) the input dictionaries and the materialization set are defined as in SMQ model. During the materialization call set sequential execution each call set is executed simultaneously and independently of one another. There is no application of any call to service allocation algorithms. Each query queue is executed independently against its service interface until it is exhausted or wanted materialization coverage is achieved.

MovieByTitle materialization of IMDB1, GM1 and GM2 is executed as three separate processes, one for each of three service interfaces, but with the same materialization storage. Each service is mapped to its own query queue, thus, making implementation of service2couple routine and consequent reseeding a straightforward 1:1 decision. Worth noting is that since the obtained materialization volume for each process is still stored in the same data storage, $distinct(r_tmp)$ function checks for duplicates of the each process against the same materialized data volume.

As stated above the distribution of calls to individual services may vary depending on the materialization properties services express during the materialization process. For instance, in an ideal situation a service interface IMDB1 with composition of materialization properties
$wd_{IMDB} \approx 0 AND ps_{IMDB} \approx maxPageSize AND rqt_{IMDB} < rqt_{si,n}$ is prioritized during the call set execution. In other words, a service that consistently returns many unique tuples in large size pages and short response times, is distributed more queries by call2service coupling than a service with high duplicate saturation or/and slow response times or/and small result set sizes.

As this is an ideal scenario not often found in the real world situation, the need arises for an orchestration of call to service allocation process. Call to service allocation process is based on the performance metrics of the services. The performance metrics are derived from the described service materialization properties. Service materialization properties are monitored during the materialization call set sequential execution. They are supplied to the call to service algorithm that in turn ranks services according to their performance

and allocates the best performer to the next set call. The call to service optimized allocation algorithms and their heuristics will be described further in the thesis.

4.6 Chapter Summary

The chapter presented service and materialization model and formally defined them. It also characterized properties related to the service and materialization models.

The chapter detailed the materialization scenarios and materialization modes of execution. Full attention was given to the definition and illustration of the materialization process.

In the next chapter we will present a case study that illustrates the importance of the data acquisition materialization dimension and related service materialization properties. It will also provide an intuition of the potential materialization process optimization via these dimensions.

Chapter 5

Strategies for Data Surfacing - Domain Coverage

5.1 Introduction

In this Chapter, as an illustration of the materialization data properties introduced in Chapter 4, we present a case study focused on a specific problem - Domain Coverage in the context of data acquisition dimension. The focus is on domain discovery and its effect on the data surfacing part of the data acquisition dimension. It presents four novel algorithms that aim to enhance the domain discovery process and improve the query coverage of the materialization process.

The study also presents query reseeding as a data surfacing strategy [Madhavan 2008]. Query reseeding is a data access method that uses available information from previous calls in order to build a materialization of maximum possible size.

The case study illustrates the importance of the data acquisition materialization dimension and related service materialization properties. It also provides an intuition of the potential materialization process optimization via these dimensions.

5.2 Case Study

In this case study we explore the influence of the materialized data properties to the outcome of the materialization process. In particular we focus on the coverage property expressed twofold as:

Coverage relative to the full materialization This property denotes the ratio between the number of items in the materialization and the number of items in the full materialization. Coverage can be further refined into a query-specific coverage that expresses a ratio relative to the portion of data in the source that satisfies the query. E.g., an application may be interested in greater coverage of data which satisfies the query "Location =Auckland".

Coverage relative to world's entities This property denotes the ratio between the number of materialized items and the number of real world items. Using multiple data sources describing the same real world entities can enhance this coverage. For instance, several services offering Auckland's "evening events" can be queried in order to produce a more comprehensive materialization. It requires duplicate elimination across different data sources.

As the access to a data source is only possible through the access limitation imposed by Access Patterns (APs), each materialization query call requires filling the AP's input fields. To achieve coverage, the domains of all legal values for such fields must be known - attribute value discovery. Such knowledge could be known in advance when, for instance, a field insists on an enumerable value set, possibly of small size (e.g., the set of movie *genres*). Otherwise, input seeding can be seen as an **incremental process** driven by the materialization queries, where the knowledge about the input fields' domains is accumulated during this process when queries are executed and output fields are retrieved.
The prior research which directly tackles the attribute domain discovery problem is sparse and the only related work proposes a random crawling technique [Raghavan 2000]. The impact of a materialization scenario on the input discovery phase can be fully appreciated by considering the two APs represented in Figure 5.1, **RealEstateByPostCode** which requires as input the real estate type - 'Rental' or 'Sale'; a Postcode and **JobByType&City** which requires as input a Job type and a City. The latter access pattern holds useful information, such as the *job title* and the *company*, but requires specific in-

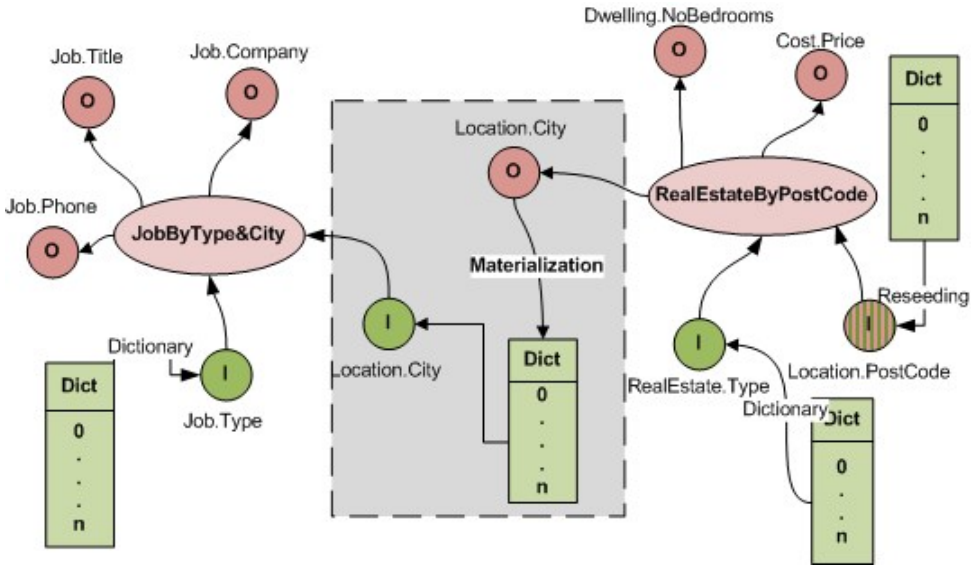


Figure 5.1: Example of the materialization problem.

put. Instead, the former service has a simple input domain, consisting of the real estate type supplied by a two-word dictionary (corresponding to 'Rental' or 'Sale') and a postcode provided by an initial dictionary and further sustained by a reseeding (self-feeding) strategy. Query reseeding strategy requires domain-matching attributes in input and output domains. In Figure 5.1 the input attribute 'postcode' is also present in **RealEstateByPostCode** result, thus, providing a base for query reseeding.

With such input, **RealEstateByPostCode** produces a list of properties described by their price, number of bedrooms, city of location and a postcode. As job types of interest are sparse, the provided list of cities via the real estate AP materialization suffices as an input provider. The materialization of **JobByType&City** iterates over all possible job types and given cities to generate the input for the mapped job search service, thereby retrieving all information about potential jobs for wanted job types and given cities.

As we can see in the example above, it is possible to query different data sources by using the same access pattern and input parameters. Since the queried data sources are independent and heterogeneous it is reasonable to expect different results even though we have queried with the same query and same access pattern.

To further illustrate the above proposition let us consider the scenario where the materialization of **JobSearch** service mapped to the **JobByType&City** AP is performed by resolving the input attribute Location.City dependency through the materialization of the three service interfaces mapped to **RealEstateByPostCode**, which are *Zillow, TradeMe* and *RealEstate.co.nz*. Each one of these services is characterized by geo-location, pagination and SLA profile characteristics as in Table 5.1, thus, influencing the materialization performance in terms of result set size and result diversity for reseeding 'post code' values. As the postcode reseeding drives the materialization of the former services this consequently ensures a steady supply of new Location.City values to the 'consuming' JobSearch service materialization process.

JobSearch(GeoLocation: NZ; SLA Profile: Daily Limit: NA; Paginated: Yes, Max Page Size: 10, Max Number of Pages: 10)	Zillow(Post Code dict (1144, 3011, 5555), GeoLocation: US; SLA Profile: Daily Limit: No limit; Paginated: Yes, Max Page Size: 100, Max Number of Pages: 10)
TradeMe(Post Code dict (1144, 3011, 5555), GeoLocation: NZ; SLA Profile: Daily Limit: No Limit; Paginated: Yes, Max Page Size: 10, Max Number of Pages: 1)	RealEstate.co.nz(Post Code dict (1144, 3011, 5555), GeoLocation: NZ; SLA Profile: Daily Limit: 1000; Paginated: Yes; Max Page Size: 20; Max Number of Pages: 10)

Table 5.1: Example SI configuration for reseeding scenario materialization with dependencies.

The materializations of the RealEstateByPostCode mapped services are initialized with $dict_{postcode}$ featuring 3 randomly selected NZ postcodes. Clearly the selection of postcodes affects the result outcome of Zillow service based in the US, as it is unlikely that NZ postcode based queries would retrieve any valid results and consequently no new postcodes values would result from the reseeding operation. Therefore its fairly generous SLA policy - No Limit - and large maxPageSize of 100 have little influence on reseeding query generation and consequent delivery of Location.City values. It is likely this service will produce the least if any Cities for the given initial dictionary.

The situation looks better for TradeMe and RealEsate services, as they are NZ based thereby making the supplied postcode dictionary more relevant. Here, we can expect some result delivery from TradeMe service but it is unfortunately restricted as its small page size limits number of newly discovered postcodes. It is likely that the queried postcode might feature in most if not all of the first 10 result tuples as top ranked results tuples are likely to be exact matches to the posed query in terms of queried value - postcode. This will consequently restrict reseeding to a very slow pace and potentially even extinguish it at an early materialization stage, thus, constricting the number of discovered Cities that JobSearch requires.

We can expect better performance from RealEstate.co.nz service as it has maxPageSize of 20 and maximum number of pages for the same query of 10. This greatly widens the number of possible new postcodes discovered as the retrieving of all chunks increases the possibility of discovering new postcodes as the total result size increases. However, even though the RealEstate service might return many new postcodes, create new queries via reseeding strategy, and, thus, provide the base for discovery of new Cities, this service is limited by its Service Level Agreement (SLA) to just 1000 queries a day. This drives the need for *prioritizing* queries based on their power to retrieve chunks with higher postcode diversity to enable us to fetch as many Location.City values before hitting the SLA imposed limit.

It becomes evident that the coverage of the Location.City domain ultimately bounds the materialization outcome of the dependent AP materialization. In the context of materialization process' query generation phase we

expect that faster exploration of the seeding domain leads to more available queries, therefore, leading to more efficient data surfacing of the materialized data source.

5.2.1 Strategies for Data Surfacing

Data surfacing involves the selection of the next query to be executed among the ones available in the *materialization queries queue*. Such a selection is performed according to a *materialization strategy*, i.e., a logic devoted to the maximization of a given set of metrics in order to optimize the query selection task. We next describe some materialization strategies for the SPSS materialization scenario, which performance is evaluated in section 5.2.4.

Let us consider a single service s described by an access pattern AP; AP has a set of input attributes I_i associated with a domain d_i, with $i = 1..n$, and a set of output attributes O_j associated with a domain d_j, with $j = 1..m$. In order to show the reseeding, we assume that $d_i = d_j$ for some i, j, i.e,. that the domain of some input and output attributes is the same. Consider $d = d_1 \times ...d_n$ as the cross product of the input domains, and let $k \in d$ be a combination of input values for the AP. A paginated query q_k^p is a query addressed to the service interface (service) si using the combination k of input values, and $1 \leq p \leq MaxNumPages$ indicates the page currently queried. We define $r_k^p \subseteq R$ as the set of tuples in the source that satisfies a query q_k^p, where R represents all the items of the source to be materialized. The input discovery step of the materialization process builds, at materialization set-up time, the initial query queue $initQ$, e.g., by retrieving them from a dictionary - $initDict$; then, new combinations of Q can be found by using the values in results r_k^p of queries that are progressively executed. The materialization R_M is built as the union of the r_k^p; note that $R_M \subseteq R$, and in general R_M is much smaller than R due to the access limitations to the data source. With a single service, the union operation is sufficient for duplicate elimination.

The outcome of a materialization strategy is influenced by the chunking of query answers, which requires multiple service calls to fully collect a query's result, and by the distribution of values for the input attributes, as distinct

inputs produce unaligned numbers of returned results, thus, introducing skew in the materialized result set. These factors call for data surfacing strategies that are capable of balancing between the coverage and diversity.

As illustrated in the example in section 5.2 issuing the queries to TradeMe service retrieves many relevant results to the query, thus, ensuring diversity but to the detriment of coverage due to the limited result set size. In contrast the RealEstate service that may return in chunks up to 2000 results per issued query. In this case some queries and its chunks might return many new cities for the queried postcode. At the same time if for instance the postcode belongs to a suburb in a major metropolitan area such a query might return results with very little variety for new cities and postcodes due to availability of many results exactly matching the posed query.

In order to define a few simple data surfacing strategies, let us model the sequence of queries produced by a data surfacing strategy as an undirected graph as described in Chapter 4. In this case the graph QRT is explored by a tree walking algorithm, where all the nodes except the root correspond to queries; the root is directly connected to queries q_k^1 with $k \in C$, and we do not further consider how nodes q_k^1 are ordered. In this context, a materialization strategy consists of interleaving of tree generation and tree traversal steps. Tree generation occurs as follows:

- If the current query q_k^p has not retrieved all the available chunks and a new chunk can be retrieved, then q_k^{p+1} is generated as a child of q_k^p,
- If the current query q_k^p has generated new query combinations h which are not present in Q, than new nodes q_h^1 are generated and Q is set to $Q \bigcup h$; the insertion of nodes q_h^1 in the tree may occur according to two insertion policies:
  - *Child insertion policy:* nodes q_h^1 are created as children (**Cq**) of q_k^p, possibly on the left of q_k^{p+1} Figure 5.2,
  - *Sibling insertion policy:* nodes q_h^1 are left-appended as children of the root, Figure 5.2.

Once new queries are appended to the tree, the materialization strategy must

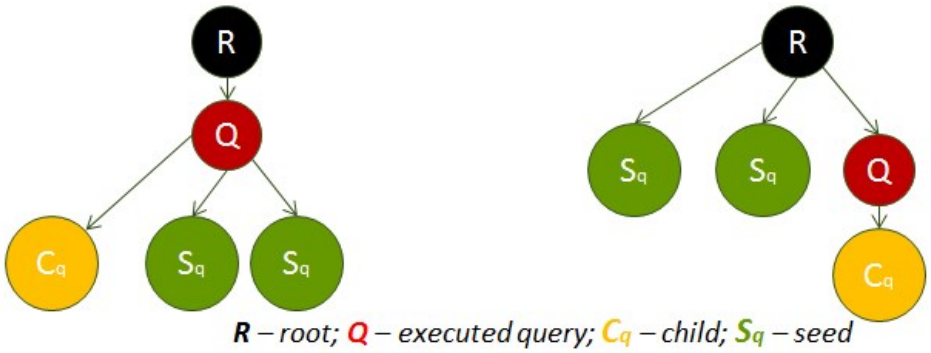


Figure 5.2: Left - Child Insertion Policy; Right - Sibling Insertion Policy.

select the next query in the tree to execute. Related works [Wu 2006] perform a similar selection process by exploiting a cost model that associates a weight to each edge in the tree, so to find an optimal selection of queries that minimizes the total cost of traversal (a *Weighted Minimum Dominating Set problem*). In this chapter, we instead exploit classical breadth-first and depth-first tree traversal algorithms. We apply them to the two variants of insertion policies, thus, obtaining four materialization strategies, which yield different performances in terms of coverage and diversity. An analysis of the performance of the proposed materialization strategies is provided in the following sections.

5.2.2 Evaluation Algorithms

In this work, we performed a preliminary study of possible materialization strategies, and we decided to address the problem from a topological point of view by exploiting the breadth-first and depth-first tree traversal algorithms. In this subsection, outlined are the **algorithms**: depth-first: "Sibling Insertion Policy" and depth-first: "Child Insertion Policy", breadth-first: "Child Insertion Policy" and breadth-first: "Sibling Insertion Policy" used in the coverage experiments described in further subsections. The algorithms are accompanied by a visualization of their propagation through the materialization process as captured by the materialization module during the materialization (domain) discovery of the experiments described in section 5.2.4. The ex-

periments were performed by considering the access pattern **RealEstateBy-Location**, which takes a real estate type and a geographical location as the input attributes. The *real estate type* input attribute was populated using a dictionary strategy, which used a static, pre-populated list of attribute values such as 'rental' or 'for sale' as the input dictionary. The location input attribute was populated by a reseeding strategy, the initial dictionary $initD$ featured set of US postcodes such as 08216, 99127, 44309. The queried web data source was paginated, max page size was set to 10.

Algorithms 1, 2, 3 and 4 detail the pseudo-code of a template for the four tree traversal algorithms. We assume a single threaded query execution model, where queries visited during the traversal of the QRT are serialized in a query queue, from which they are fetched one at a time for execution. The algorithm template consists of three phases: materialization initialization, allocation of reseeded queries in the buffer, and allocation of next page queries in the buffer. The $createQ$ function is responsible for the query generation step which, given the currently discovered field domain values $initD$, creates the set of queries that are added to the execution queue: at initialization time, queries are added by the enqueue function; at query reseeding time, instead, reseeded queries are added using the $insertReseededQuery$ function, while next-page queries are added with the $insertNextQuery$ function.

Given a currently processed query, depth-first algorithms push next page queries at the head of the query queue, thus, assuring that the result set associated with the current input combinations is exhausted before probing new combinations; in this context, the Sibling Insertion policy imposes the new query to be appended at the bottom of the queue, thus, deferring the evaluation of the new input combinations after all the previously discovered ones; the Child insertion policy, instead, inserts the new query right after the currently executed one, thus, assuring that it will be executed as soon as the current input combinations are exhausted.

In breadth-first strategies are, instead, more involved. When the Child insertion policy is applied, newly discovered queries are added to query queue as a child of the currently processed query, while the next page of the currently processed query (if it exists) is placed at the bottom of the query queue.

The application of the sibling insertion policy requires a more articulated query serialization algorithm, and it is depicted in 3. We assume the query queue to be n-dimensional, where the number of dimensions (levels, in the algorithm description) is bounded by the maximum number of pages that can be retrieved for a given query. A level, therefore, represents a query page number. Newly discovered queries are added as the last in the current level, while the new page of the currently processed query (if it exists) is placed at the bottom of the next level. Queries are fetched execution-level-wise, i.e., page-2 queries are fetched only when all the page-1 queries have been executed.

The algorithms are accompanied by the **visualization** of their propagation through the materialization process as captured by the materialization module during the materialization discovery.

Depth-first "Sibling insertion policy" - newly discovered seed is added to the query queue as a child of its parent (currently processed query), while the next page of the currently processed query (if exists) is placed at the head of the queue; currently processed query is removed.

Algorithm 1 Depth-First: "Sibling Insertion Policy"

$R_M \leftarrow \varnothing$ //empty materialization
$currDict \leftarrow initD$ //current dictionary initialized
$initQSet \leftarrow createQ(currDict)$ //initial query set created
$QRT \leftarrow populate(initQSet)$ //graph populated
$queryQueue \leftarrow enqueue(QRT)$ //query queue as QRT traversal
while *QueryQueue* is not empty **do**
  $q \leftarrow getHead(queryQueue)$
  $d(q), r_k^p \leftarrow execute(q)$
  $currDict \leftarrow D(q)$
  $R_M \leftarrow R_M \cup r_k^p$
  $seededQSet \leftarrow createQ(currD^n)$
  while seededQSet is not empty **do**
    $sq \leftarrow qetHead(seededQSet)$ // sq - seeded query
    if sq not in queryQueue AND sq not in $executedQueries$ **then**
      $queryQueue \leftarrow insertReseededQuery(nextPage(q))$
    end if
  end while
  if hasNextPage(q) **then**
    $queryQueue \leftarrow insertNextQuery(nextPage(q))$
  end if
  $queryQueue \leftarrow queryQueue\ q$
  $executedQueries \leftarrow executedQueries \cup q$
end while

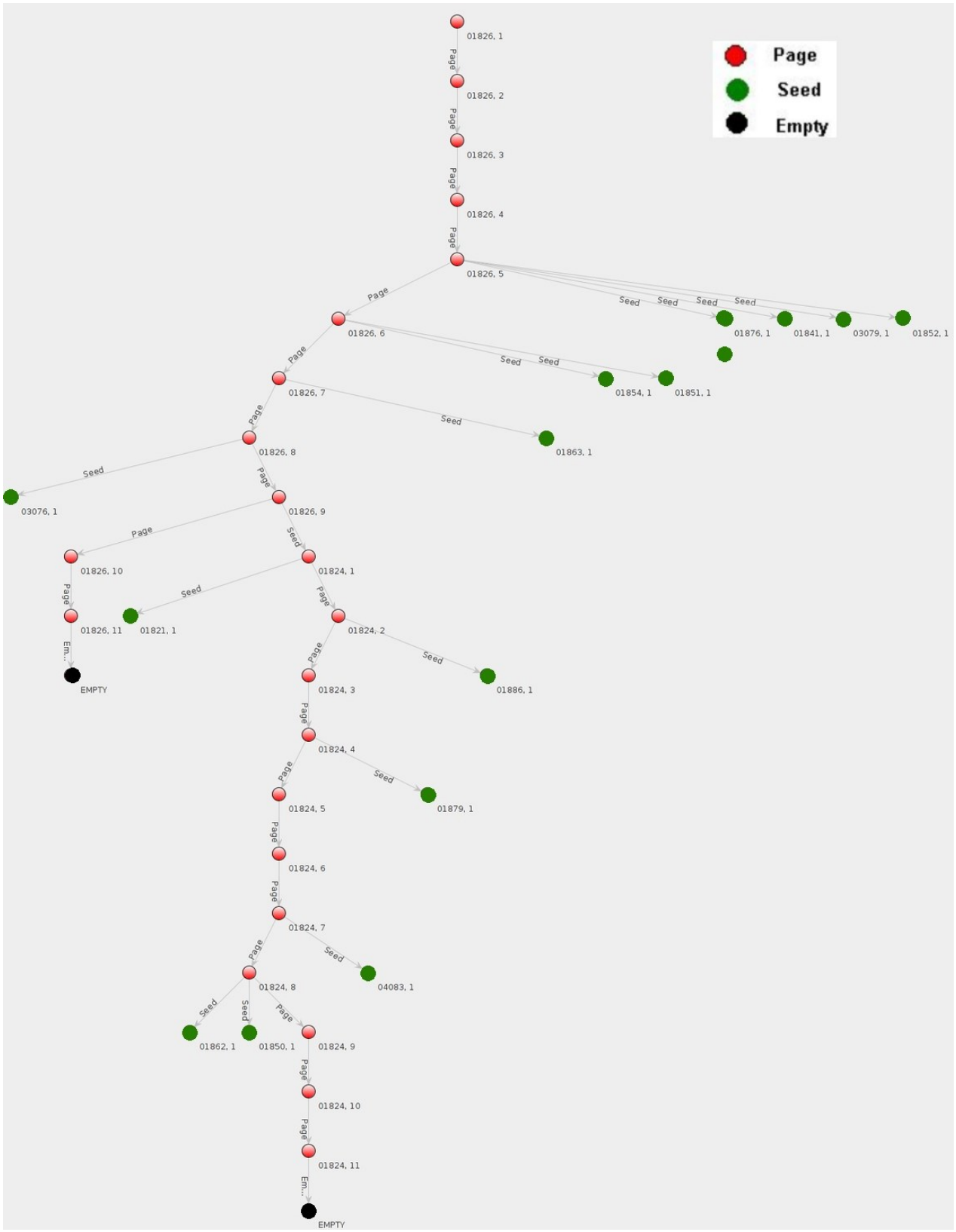


Figure 5.3: Depth-first: "Sibling Insertion Policy" run-time visualization.

Depth-first *"Sibling insertion policy"* visualization example in Figure 5.3 illustrated the algorithm behaviour. For the given query with postcode 01826 results pages 1-10 have returned non empty result sets with pages 5, 6, 7, 8 and 9 discovering new seeds (new postcodes). Following the algorithm behaviour once the 11^{th} page was returned empty, the query 01826 was interpreted as exhausted and removed from the query queue. The algorithm retracted to the first available - last discovered seed at the top of the queue which happened to be seed 01824, discovered in page 9, that became a new query queue head, and then continued.

Depth-first "Child Insertion Policy" - newly discovered seed is added at the queue bottom, while the next page of the currently processed query (if it exists) is placed at the head of the query queue; currently processed query is removed.

Algorithm 2 Depth-First: "Child Insertion Policy"

$R_M \leftarrow \varnothing$ //empty materialization
$currDict \leftarrow initD$ //current dictionary initialized
$initQSet \leftarrow createQ(currDict)$ //initial query set created
$QRT \leftarrow populate(initQSet)$ //graph populated
$queryQueue \leftarrow enqueue(QRT)$ //query queue as QRT traversal
while *QueryQueue* is not empty **do**
  $q \leftarrow getHead(queryQueue)$
  $d(q), r_k^p \leftarrow execute(q)$
  $currDict \leftarrow D(q)$
  $R_M \leftarrow R_M \cup r_k^p$
  $seededQSet \leftarrow createQ(currD^n)$
  while seededQSet is not empty **do**
    $sq \leftarrow poll(seededQSet)$ // sq - seeded query
    if sq not in queryQueue AND sq not in $executedQueries$ **then**
      $queryQueue \leftarrow addAtPosition(getPosition(q), sq)$
    end if
  end while
  if hasNextPage(q) **then**
    $queryQueue \leftarrow addFirst(nextPage(q))$
  end if
  $queryQueue \leftarrow queryQueue\ q$
  $executedQueries \leftarrow executedQueries \cup q$
end while

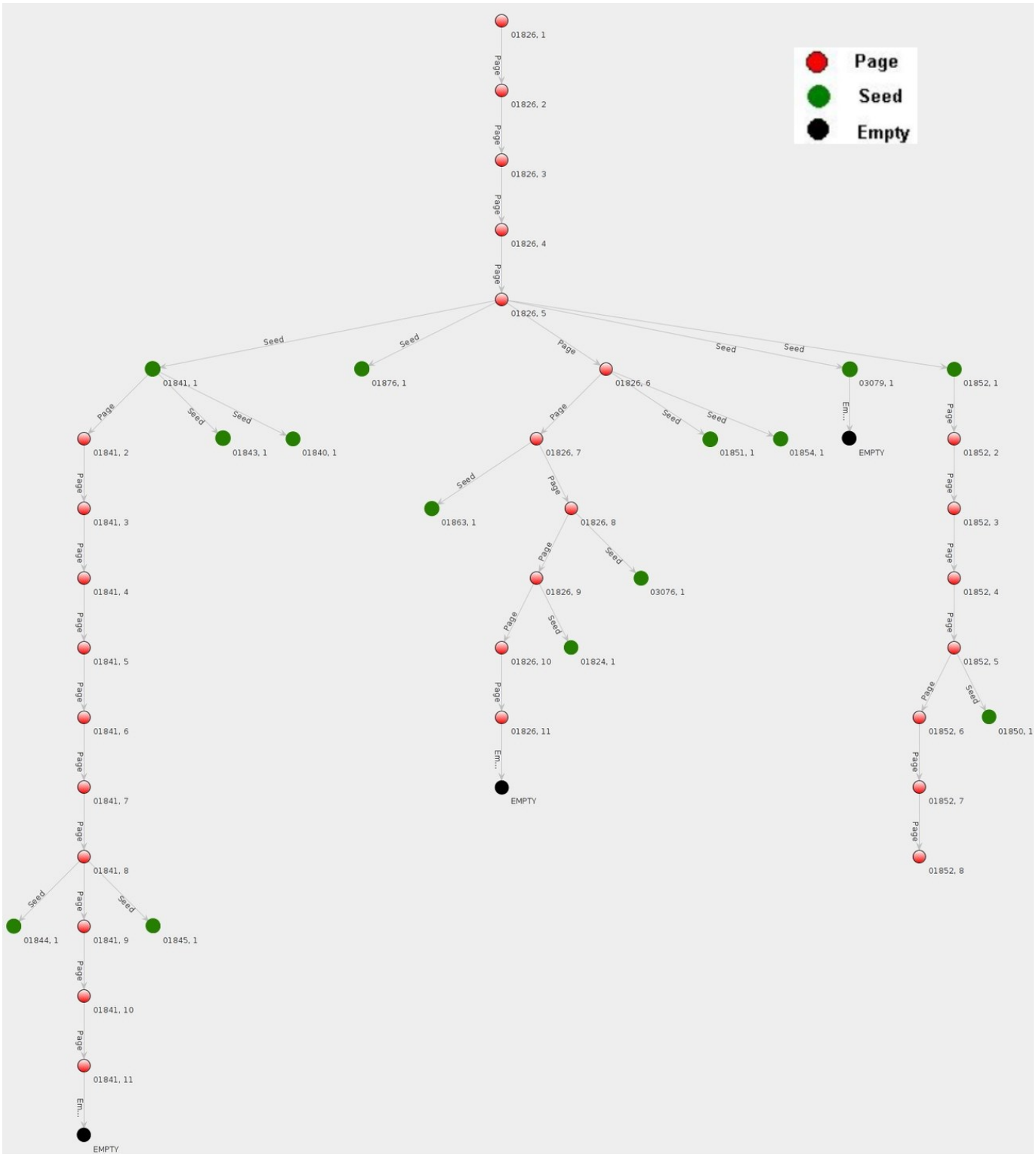


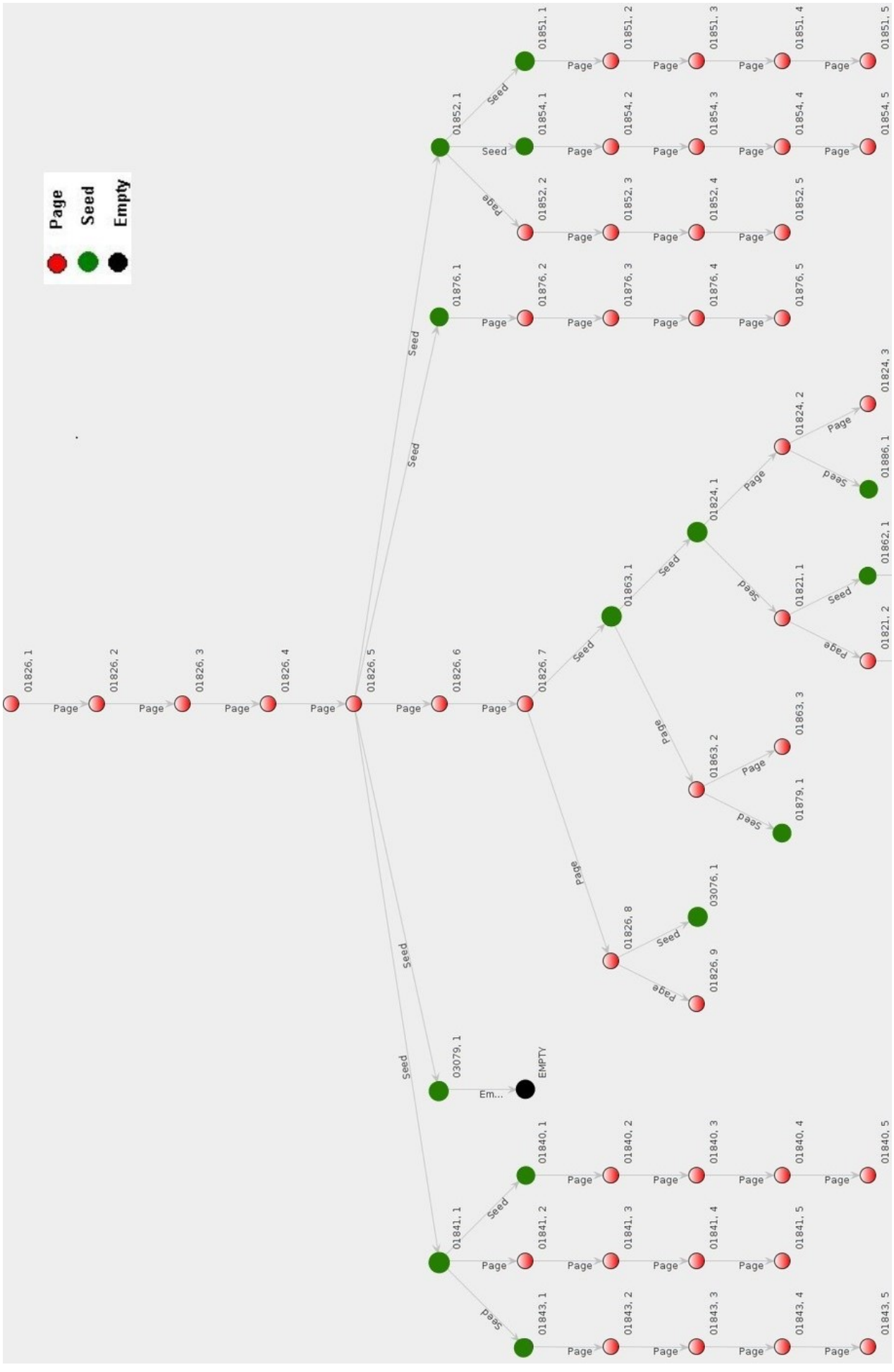
Figure 5.4: Depth-first: "Child Insertion Policy" run-time visualization.

Depth-first *"Child Insertion Policy"* run-time visualization example in Figure 5.4 illustrated the algorithm behaviour. For the given query with postcode 01826 results pages 1-10 have returned non empty result sets with pages 5, 6, 7, 8 and 9 discovering new seeds (new postcodes). Following the algorithm behaviour once 11th page was returned empty, the query 01826 was interpreted as exhausted and removed from the query queue. The algorithm retracted to the first available seed 03079 discovered in page 5, that became a new query queue head, and continued. Query 03079 didn't produce any new seeds so once exhausted the new query queue head became 01852 post code discovered in page 5 of 01826 and continued.

Breadth-first "Sibling Insertion Policy" - newly discovered seed is added as the last in the current level, while the new page of the currently processed query (if it exists) is placed at the bottom of the next level; currently processed query is removed. (Worth noting is that this version of the breadth-first version is spatially organized into two levels of complexity; upon exhaustion the top (first) level is removed while the bottom (second) level is promoted as first and the new (empty) second level added to the data structure.)

Algorithm 3 Breadth-First: "Sibling Insertion Policy"

$R_M \leftarrow \varnothing$ //empty materialization
$currDict \leftarrow initD$ //current dictionary initialized
$initQSet \leftarrow createQ(currDict)$ //initial query set created
$QRT \leftarrow populate(initQSet)$ //graph populated
$queryQueue \leftarrow enqueue(QRT)$ //query queue as QRT traversal
while $QueryQueue$ is not empty **do**
  $q \leftarrow getHead(queryQueue)$
  $d(q), r_k^p \leftarrow execute(q)$
  $currDict \leftarrow D(q)$
  $R_M \leftarrow R_M \cup r_k^p$
  $seededQSet \leftarrow createQ(currD^n)$
  while seededQSet is not empty **do**
    $sq \leftarrow getHead(seededQSet)$ // sq - seeded query
    if sq not in queryQueue AND sq not in $executedQueries$ **then**
      $queryQueue \leftarrow addLastCurrentLevel(sq)$
    end if
  end while
  if hasNextPage(q) **then**
    $queryQueue \leftarrow addLastNextLevel(nextPage(q))$
  end if
  $queryQueue \leftarrow queryQueue\ q$
  $executedQueries \leftarrow executedQueries \cup q$
  if currentLevel is empty **then**
    $currentLevel = nextLevel$
    $nextLevel = addNewLevel()$
  end if
end while


Page
Seed
Empty
01826, 1
01826, 2
01826, 3
01826, 4
01826, 5
01826, 6
01826, 7
01826, 8
01826, 9
01851, 1
01851, 2
01851, 3
01851, 4
01851, 5
01852, 1
01852, 2
01852, 3
01852, 4
01852, 5
01854, 1
01854, 2
01854, 3
01854, 4
01854, 5
01876, 1
01876, 2
01876, 3
01876, 4
01876, 5
01824, 1
01824, 2
01824, 3
01886, 1
01863, 1
01863, 2
01863, 3
01879, 1
01821, 1
01821, 2
01862, 1
03076, 1
03079, 1
EMPTY
01840, 1
01840, 2
01840, 3
01840, 4
01840, 5
01841, 1
01841, 2
01841, 3
01841, 4
01841, 5
01843, 1
01843, 2
01843, 3
01843, 4
01843, 5

Breadth-first *"Sibling Insertion Policy"* run-time visualization example in Figure 5.5 illustrated the algorithm behaviour. For the given query with postcode 01826 results pages 1-10 have returned non empty result sets with pages 5, 6, 7, 8 and 9 discovering new seeds (new postcodes). Following the algorithm behaviour once the new seeds were discovered in page 5, page 5 was placed at the bottom of the next level in the queue and a new query's page 1 with value 01876 was executed until it returned no new seeds. The process then retracted to seed 01852 discovered by the page 5 of the first query 01826. Postcode 01852 page 1 discovered new seeds, 01852; page 2 was put at the bottom of the next level and another seed 01854 continued execution, upon its exhaustion (no new seeds discovered) process switched back to seed of 01852, which happened to be 01851, and then continued until it was exhausted - no new seeds. The process continued with the first query 01852 seed corresponding to page 5 - query 01841.

Breadth-first "Child Insertion Policy" - a newly discovered seed is added to the query queue as a child of its parent (currently processed query), while the next page of the currently processed query (if it exists) is placed at the bottom of the query queue; currently processed query is then removed.

Algorithm 4 Breadth-First: "Child Insertion Policy"

$R_M \leftarrow \varnothing$ //empty materialization
$currDict \leftarrow initD$ //current dictionary initialized
$initQSet \leftarrow createQ(currDict)$ //initial query set created
$QRT \leftarrow populate(initQSet)$ //graph populated
$queryQueue \leftarrow enqueue(QRT)$ //query queue as QRT traversal
while *QueryQueue* is not empty **do**
  $q \leftarrow getHead(queryQueue)$
  $d(q), r_k^p \leftarrow execute(q)$
  $currDict \leftarrow D(q)$
  $R_M \leftarrow R_M \cup r_k^p$
  $seededQSet \leftarrow createQ(currD^n)$
  while seededQSet is not empty **do**
    $sq \leftarrow poll(seededQSet)$ // sq - seeded query
    if sq not in queryQueue AND sq not in $executedQueries$ **then**
      $queryQueue \leftarrow addAtPosition(getPosition(q), sq)$
    end if
  end while
  if hasNextPage(q) **then**
    $queryQueue \leftarrow addLast(nextPage(q))$
  end if
  $queryQueue \leftarrow queryQueue\ q$
  $executedQueries \leftarrow executedQueries \cup q$
end while

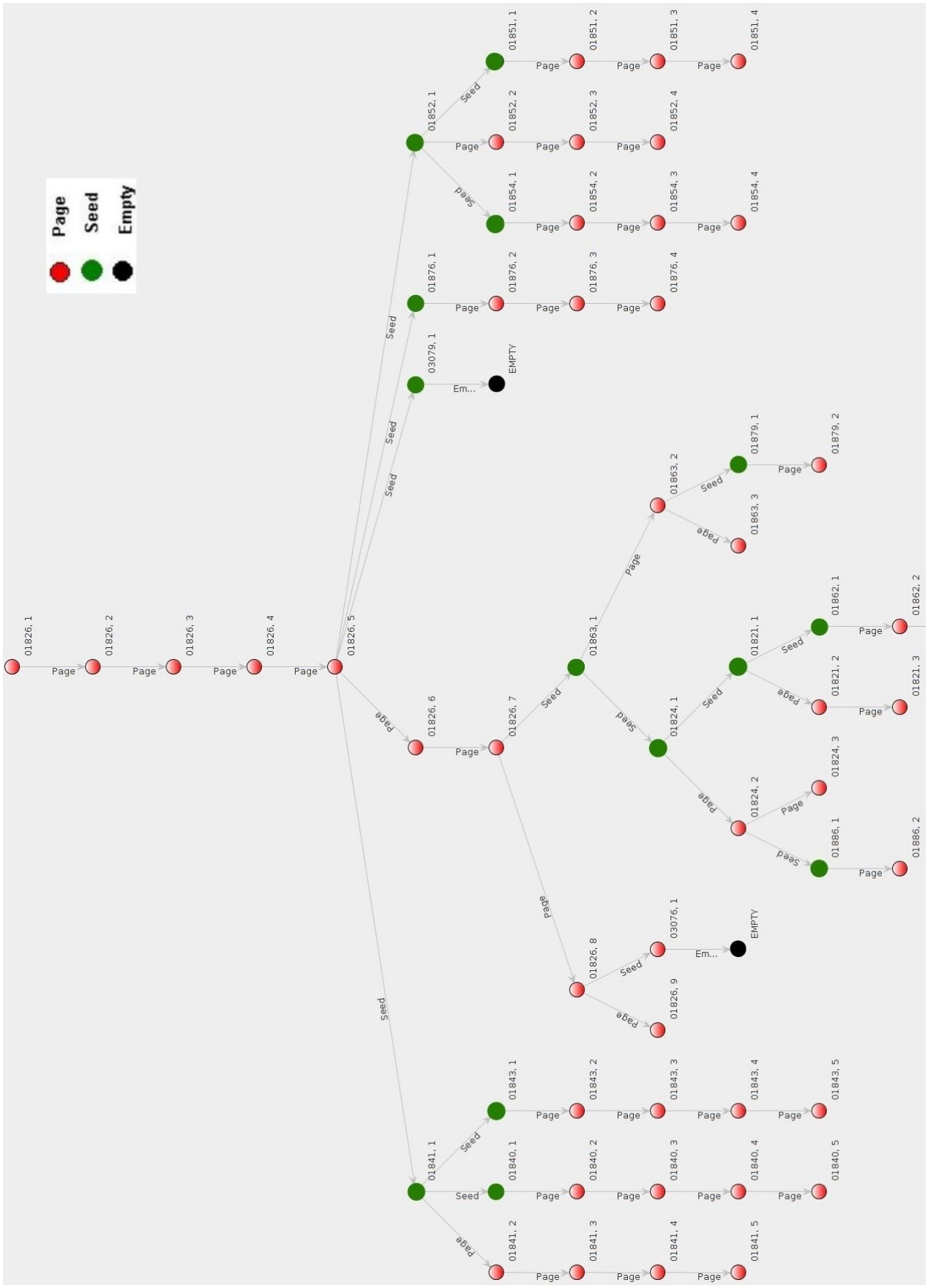


Figure 5.6: Breadth-first: "Child Insertion Policy" run-time visualization.

Breadth-first *"Child Insertion Policy"* run-time visualization example in Figure 5.6 illustrated the algorithm behaviour. For the given query with postcode 01826 results pages 1-10 have returned non empty result sets with pages 5, 6, 7, 8 and 9 discovering new seeds (new postcodes). Following the algorithm behaviour once the new seeds were discovered in page 5, it was placed at the bottom of the queue and new query 03079 page 1 was executed resulting in an empty result set then a new query, seed 01876, pages 1, 2, 3 were executed until exhausted. As they did not discover new seeds, the process then retracted to seed 01852 discovered by the first query 01826 page 5. Postcode 01852 page 1 discovered new seeds, 01852 page 2 was put at the bottom of the next level and another seed 01854 continued execution, upon its exhaustion (no new seeds discovered) the process switched back to seed of 01852, which happened to be 01851 and continued until it was exhausted - no new seeds. The process continued with the first query 01826 seed of its page 5 - query 01841.

5.2.3 Materialization Module Architecture

In order to support the materialization process, for the purposes of this work we have designed and developed a materializer module that sits within the broader SeCo framework [Bozzon 2011a] - Figure 5.7. The module relies on the descriptions stored with the existing SeCo service mart repository, and utilizes the existing SeCo QP (query processor) API implementation. The materialization module is comprised of:

- Materialization Profiles repository, which contains the service and materialization properties of the Service Description Framework Service Interfaces and Access Patterns,
- Data Acquisition module, which implements the input discovery strategies (e.g., dictionary, reseeding and query log data acquisition), guided by the data model. Implemented:
    - Dictionary Strategy - an existing dictionary of relevant input terms is used to populate the input attributes in the queries,

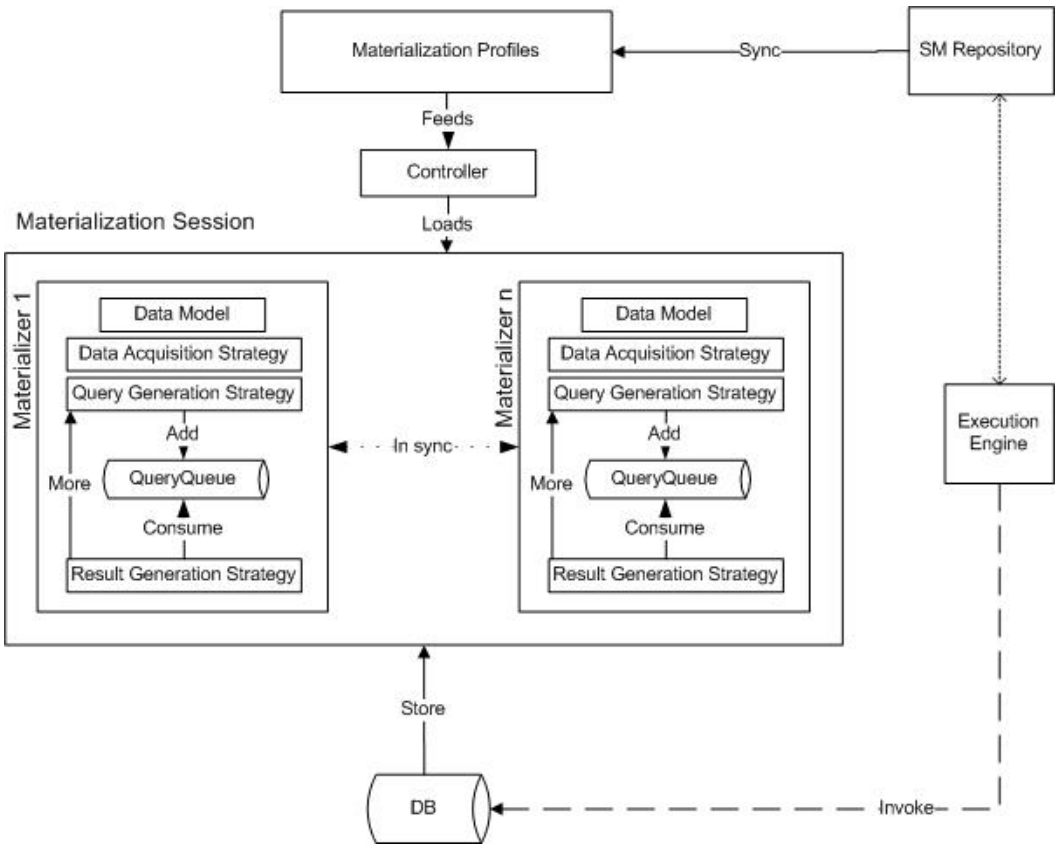


Figure 5.7: Architecture of the materialization module.

- Reseeding Strategy - the input attribute values are selected from the results obtained by the set of initial 'seeding' queries and consequent queries in recursive fashion until all result inputs are exhausted,
- Consumer Strategy - input attribute values are 'consumed' from the producer queue as they become available,
- Reseeding Consumer Strategy - input attribute values are obtained as a combined input from Cosumer and Reseeding Strategy,
- Random From DB Strategy - input attribute values are obtained as a random selection from available DB containing input attribute domain dictionary.

- Query Generation module, which generates the queries to be executed and adds them to a data structure PQ representing the pending queries to be executed. Implemented:
  - Cartesian Product - in cases where there is more than one input attribute, queries are generated by creating all pair-wise combinations of the given input values,
  - Cartesian Product Scored - in cases where there is more than one input attribute, queries are generated by creating all pair-wise combinations of the given input values and returned as a queue ordered by the supplied score function - frequency, harvest rate, diversification etc.,
  - Cartesian Product Centrality Scored Strategy - in cases where there is more than one input attribute, queries are generated by creating all pair-wise combinations of the given input values and ordered by the centrality betweenness function against one of the input attribute domain,
  - Twitter Query Generation Strategy - query generation strategy specific to the requirement of the Twitter materialization procedure. Twitter materialization procedure enables materialization of Twitter messages for provided hash tag dictionary via Twitter API

[Twitter 2014]. The query generation is self-driven by reseeding via hash tags obtained from retrieved tweets.

- Result Generation Module, which extracts the next query from the query queue and launches its execution. Implemented:
  - Result Generator Breadth_1 - Breadth first - child insertion policy,
  - Result Generator Breadth_2 - Breadth first - sibling insertion policy,
  - Result Generator Depth_1 - Depth first - child insertion policy,
  - Result Generator Depth_2 - Depth first - sibling insertion policy,
  - Result Generator Graph - result generation strategy driven by the supplied graph algorithm,
  - Result Generator Random - result generation strategy that randomly extracts queries from the query queue.
- Materialization Analytics module, which processes discovered materialization data in run time and computes the result set and output domains based statistics. Implemented:
  - Domain Coverage Analytics - analysis coverage of the domain space in run-time,
  - Result Coverage Analytics - analysis total result coverage against the pre-materialized data,
  - Centrality Betweenness Analytics - computes a number of shortest paths from all vertices to all others that pass through the discovered domain values,
  - Domain Connectivity Analytics - determines the connectivity of the values in the domain as they are discovered,
  - Frequency Analytics - computes number of occurrences of the reseeding value in the materialization during materialization process,

- Harvest Rate Analytics - computes a number of results sets containing the reseeding value during materialization process,
- Diversification Analytics - computes diversification rate of the reseeding value during materialization process.

A single materialization is driven by a materialization controller, which is in charge of determining the order of execution for the queries produced during the Query Generation step, by traversing the data structure of pending queues according to a given materialization strategy.

Two additional components take care of the Multi Pattern Multi Service expansion of the model. Materialization Session Controller, which builds an initial plan of the MPMS materialization by performing the reachability analysis. It monitors the service interface performance statistics dynamically adjusting the materialization execution process. Materialization execution session, which maintains the execution environment of the plan, i.e., encapsulates all the individual materializations as traverse down the tree of execution while enforcing synchronization between materialization data structures.

5.2.4 Evaluation Results

5.2.4.1 Domain Coverage

We evaluated the efficiency of the materialization strategies described in Section 5.2.1 in terms of **domain coverage** for each attribute with respect to the number of queries required to achieve a given coverage value. The goal is to assess the ability of each materialization strategy to quickly explore the data and domain-space of a data source. To perform the evaluation, we created a database composed of $\tilde{1}$M real estate offers crawled from an existing Real Estate Web site. Experiments were performed by considering the access pattern **realEstateByLocation**, which takes a real estate type and a geographical location as the input attributes.

The *real estate type* input attribute was populated using a dictionary strategy, which used a static, pre-populated list of attribute values such as 'rental' or 'for sale' as the input dictionary. The location input attribute was populated

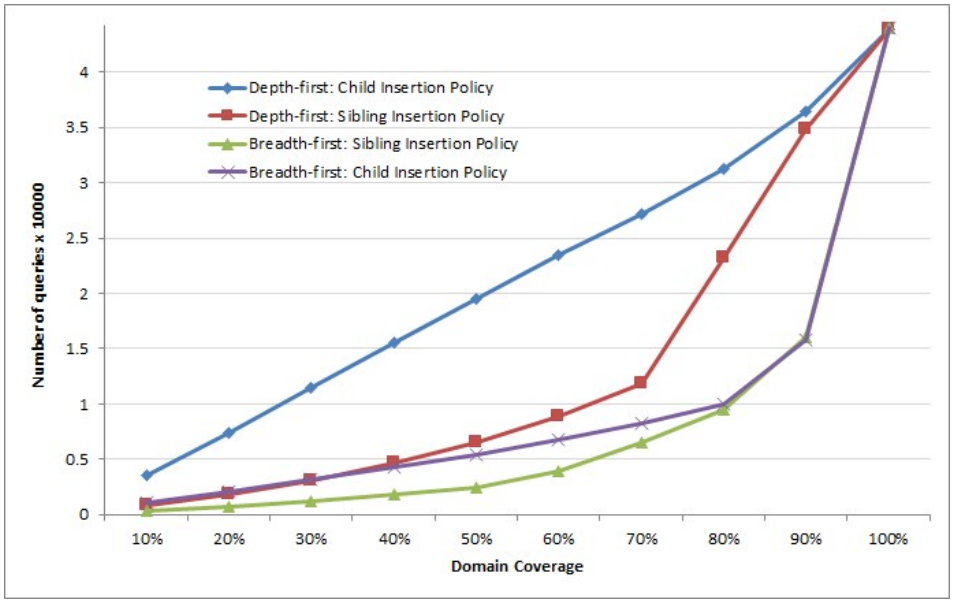


Figure 5.8: Experimental Results for SPSS domain coverage.

by reseeding strategy; moreover a matching location output attribute returning locations was mapped as an appropriate input attribute value provider. A domain of postcodes found in the collected real estate database was used as the location input attribute domain. The domain size of the postcodes reseeding input attribute was approximately 11000, a randomly selected subset of which was used as a starting seed dictionary.

For each of the materialization (result generation) strategies 10 runs were performed; to avoid biases in the evaluation, the input attributes conforming to the reseeding input strategy were initialized at each run by a randomly selected subset *initDict* of size 100. The resulting domain coverage increase had been averaged between runs for each strategy and observed in 10% increments (with regards to the overall domain size).

5.2.4.2 Coverage relative to the full materialization

We evaluated the efficiency of the materialization SPSS strategy described in the previous subsection in terms of result coverage for each query with regards to the number of queries required to achieve a given coverage value. The setup of the experiment was the same as in the domain coverage experi-

ment. We used a database composed of ~1M real estate offers crawled from an existing Real Estate Web site. Experiments were performed by considering the access pattern realEstateByLocation, which takes a real estate type and a geographical location as the input attributes. While obtaining the origi-

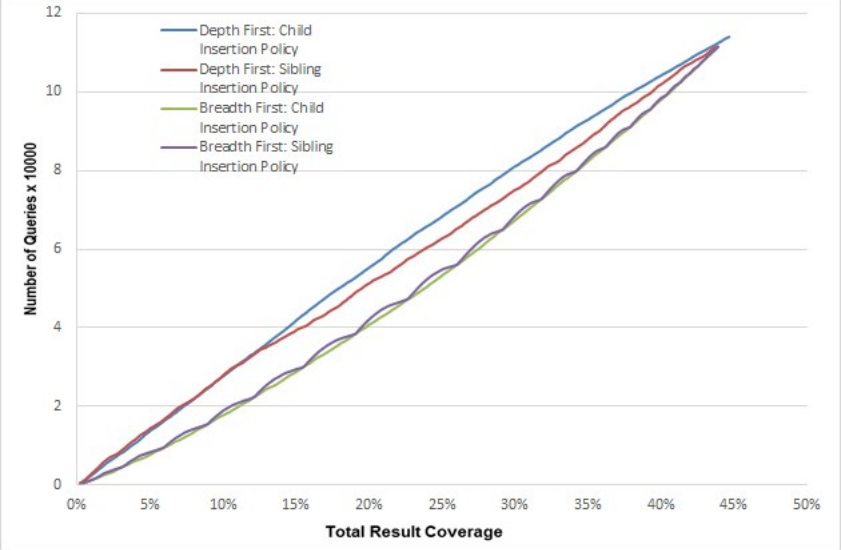


Figure 5.9: Experimental Results SPSS - Total Result Coverage - page size 10.

nal materialization the set of queries used for the discovery was serialized in a DBMS. During the experiment a subset of 80% randomly selected queries of the original query set were executed in batches of 500 against the original materialization. After each batch the difference between the experiment materialization data and the original materialization was taken. Two sets of experiments were performed each one with 10 runs. First experiment was performed with chunk size 10 - minimum page size, while the other experiment was performed with chunk size 100 - maximum page size. The resulting result coverage increase had been averaged between runs for each strategy and observed in 10% increments (with regards to the overall result size).

5.2.4.3 Evaluation Results Discussion

We compared the performance of Breadth First Search (BFS) and Depth First Search (DFS) algorithms, each featuring two different child/sibling insertion

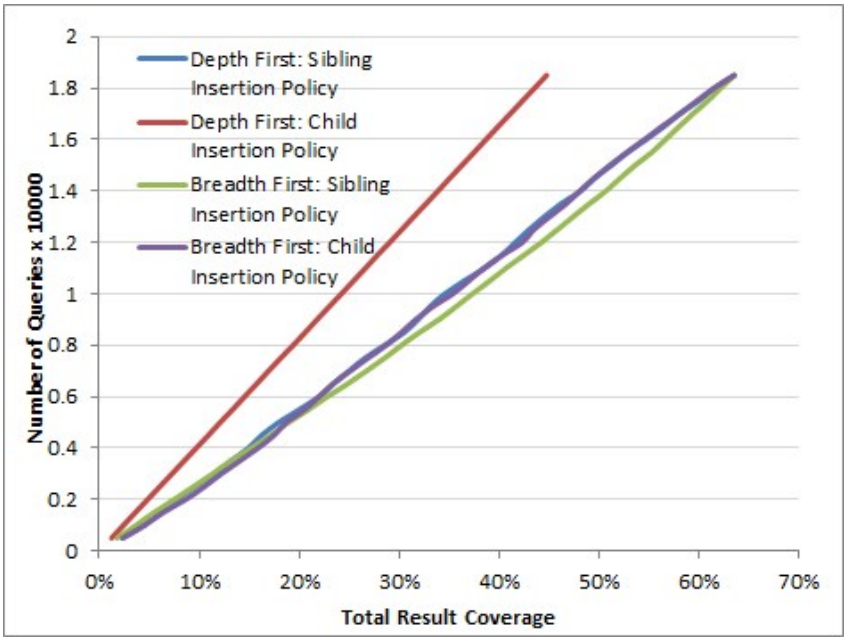


Figure 5.10: Experimental Results SPSS - Total Result Coverage - page size 100.

policies. The results were obtained against a database of 1M real estate offers crawled from several existing Real Estate Web sites. Figure 5.8 depicts the change of coverage in respect to the number of issued queries.

We also performed the real estate database coverage experiments with varying result set sizes. Figures 5.9 and 5.10 depict the change of coverage in respect to the number of issued queries when result set size was set to 10 and 100 results tuples respectively.

Figure 5.8 provides the results of our comparison. Breadth-first algorithms are able to retrieve a wider coverage on the input by requiring less service invocations. For the tested scenario, the first breadth-first algorithm is able to achieve a 65% coverage of the input domain after 5000 queries (values in line with the daily limit imposed by several Web data source API providers), thus, requiring 25% less queries than the second best breadth-first algorithm and 50% less queries than the best performing depth-first algorithm and almost a 5-fold improvement over the worst, depth-first based algorithm. The depth-first algorithm, instead, proves very unsuited for the goal at hand. The experimental results match our intuition.

The variance in the performance between BFS and DFS algorithms matches our intuition. DFS algorithms focus on executing all pages of the given query attribute value, thus, narrowing the scope of the potential domain discovery. The queried attribute value may suffer from poor connectivity within the domain. Another reason might be the feature of the query whereas r_k matched top-k tuples of the query, thus, restricting the new domain value discovery. Conversely we observed the situation where the query is too specific to return any values.

In conclusion, if the query is neither in top-k range nor too exact, the discovery of domain values of an attribute with unknown domain is enabled. Contrasting to DFS, BFS switches to queries with new values as soon as discovered, thus, widening the discovery scope. The behaviour also limits the negative influence of poorly connected values to the discovery rate. BFS algorithm is still suspect to the value discovery optimal situation where the query is neither in top-k nor exact as elaborated above.

Results presented in this study case match work done by [Jin 2011]. Here authors employed breadth and depth search algorithms alongside their improved algorithms against several hidden web databases to ascertain their domain value discovery performance.

Figure 5.9 provides the results of our second comparison. The results of both result set size experiments match the behaviour of the strategies observed in the domain coverage experiments. Breadth-first algorithms are able to retrieve a wider coverage on the input by requiring less service invocations. For the tested scenario, the first breadth-first algorithm is able to achieve a 25% coverage of the total result domain after 46000 queries, thus, requiring 18% less queries than the second best breadth-first algorithm and 25% less queries than the best performing depth-first algorithm and almost a 3-fold improvement over the worst, depth-first based algorithm.

Noticeable is the drastic difference in the number of executed queries to achieve the full result, in case of the first experiments with chunk size 10 - 120000 queries were issued while in the case of second experiments with chunk size 100, just 18000 queries were issued, almost a 9-fold decrease. This result correlates research results presented in work by [Wu 2006], where sim-

ilar decrease in result size resulted in 50-60% drop of the domain discovery performance.

In summary the case study case illustrates the importance of the engagement of the appropriate domain value discovery algorithm. Failure to perform in this phase of the materialization is likely to result in, from the number of used queries, an expansive materialization procedure. This also increases the risk of the materialization process not achieving the desired result as data acquisition phase may fail to discover new values for all the possible queries needed to achieve the assigned task.

5.3 Chapter Summary

In the previous two chapters we presented service and materialization models and formally defined them.
This chapter brought a case study that illustrated input discovery as a challenge within the materialization data acquisition dimension. The case study presents our approach and experiment results proving the validity of this approach. It reflects on the related research, it outlines similarities and findings that are to be proved of essence in the materialization process optimization. In this chapter we also described the architecture of the materialization module used in the empirical study throughout the research.

In the next chapter we will address the first materialization formulation step - finding a feasible solution for the materialization task from the given set of access patterns and mapped services.

Chapter 6

Modelling Feasible Solutions

6.1 Introduction

In this Chapter we focus on materialization feasibility analysis. Feasibility analysis presents a critical part of the formulation of the materialization solution as it delivers a combination of access patterns that meet required coverage requirements.

The service materialization formulation and its feasibility analysis are expressed in terms of Service Description Framework and Service materialization model as presented in Chapter 4.

The analysis of feasibility determines which combination of access patterns produces wanted materialization considering the available input dictionaries. It also provides an estimate of the materialization coverage by determining a maximum number of materialization calls, or boundedness, for the given feasible solution.

We present the transformation of the SDF into a network layout from which the feasibility model is derived and analysis performed upon. We formally define the feasibly analysis model and its relation to the service materialization model. Then we elaborate on the actual feasibility study that leads to the feasibility analysis model. On top of this, the model is presented as a working algorithm to be extended in the materialization process. Finally, to evaluate the effectiveness of the feasibility analysis we experiment with a set of materialization tasks of varying Access Pattern complexity.

6.2 Service Materialization Feasibility Model

The service materialization feasibility model defines concepts novel to the SDF and web sources materialization in general. It also expands and complements some of the SDF definitions. The logical level of the SeCo Service description framework may contain many access patterns with overlapping input and output attributes domains. The analysis of feasibility within the selected set of access patterns is necessary in order to select the combination of access patterns which when executed produces a materialization output. The analysis of feasibility considers a materialization task by taking a set of access patterns from SDF as input as well as a set of input dictionaries and determines which combination of access patterns for the provided dictionary is able to achieve a materialization task. This is also referred as a *reachability* of the proposed solution. The analysis further defines *boundedness* of the feasible solution by taking reachable solution and input dictionaries as input and considering all attributes in the output domains which value delivering power and position in respect to input attributes restrict the number of materialization calls. An

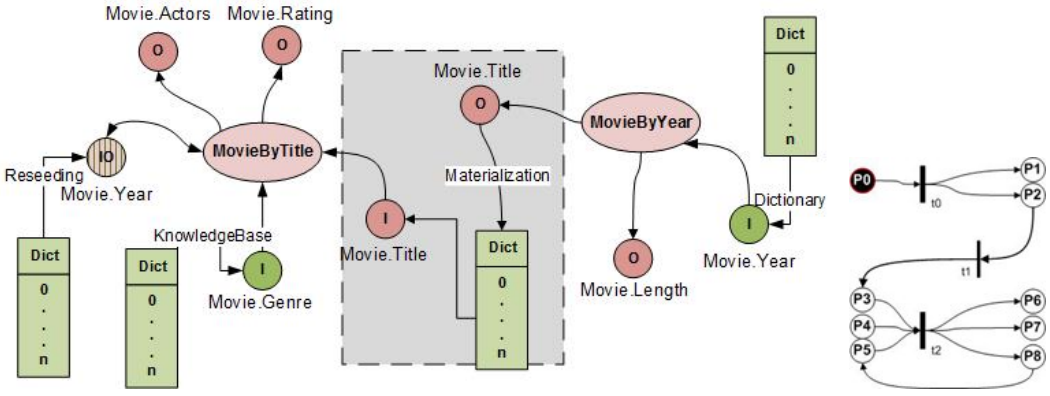


Figure 6.1: Example of schema of the materialization problem at AP level and its Petri Net representation.

access to a data source is shaped by the access limitations imposed by Access Patterns (APs). Each materialization query call requires filling the AP's input fields. To maximize coverage, the domains of all legal values for such fields must be known. Such knowledge could be known in advance when, for instance, a field insists on an enumerable value set, possibly of small size

(e.g., the set of movie genres). Alternatively, input seeding can be seen as an incremental process driven by the materialization queries, whereby the knowledge about the input field domains is accumulated during this process when queries are executed and output fields are retrieved. As shown in Figure 6.1 the MovieByYear materialization reseeds MovieByTitle via the connection pattern Movie.Title.

We illustrate the impact of a materialization scenario on the input discovery phase by considering the two APs represented by MovieByTitle (which requires as input the Movie's Title, Genre, and Year) and MovieByYear (which requires as input the Movies' Year). The former AP has useful information, such as Actors and the Rating, but requires very specific input. Instead, the latter AP has a simple input domain, consisting of the Year of issue (e.g., 2011); with such input, it can produce a list of titles. Movie Genres are few, and thus an input generator that already knows the Title and is set on the current Year can iterate over all possible genres to generate the input for the former service, thus extracting all the information about actors and ratings of current Year movies.

The described AP interaction is further represented in a Petri net model [Murata 1989]. In the proposed Petri net model input values are tokens that propagate through the net whose nodes (places) reassemble the input and output domain attributes. In the Petri net model the input domain is on the left side of the net while the output domain is on the right side. It is assumed that every input attribute is supplied by tokens from its own input dictionary of values or from the equivalent output domain attribute from the same access pattern (reseeding) or from another access pattern. The token propagates from left side to the right, driven by the firing of the query, and in some situations from the right side to the left if there is such a connection between access patterns. The edges are the arcs between the input and output domains of the access patterns. The propagation of tokens through the net is triggered by issuing of queries called transactions in the Petri net model.

Following this representation the above scenario translates to the sequence of token moves from P0 (Movie.Year), triggered by t0 (query) to P1 and P2 (Movie.Length and Movie.Title). The token from P2 propagates to P3

thereby supplying Movie.Title value from the output domain of MovieByYear to the input domain of the dependant MovieByTitle access pattern. The MovieByTitle access pattern executes if there are tokens in each place of its input domain, provided that there is an input dictionary attached to P4 and P5 supplying tokens, and if the token was supplied by P2 to P3 (see Figure 6.1).

By analysing token propagation through the net we deduce the *reachability* of the proposed access pattern combination. In the presented scenario the tokens will propagate through all places of the net if and only if there is an input dictionary with at least one value for places P0, P4 and P5 and if the top AP query execution (MovieByYear) resulted in a value at P2. Furthermore, by analysing the size of the input dictionaries and the size of the discovered output attribute domain we measure the *boundedness* of the proposed access pattern. The number of materialization calls for the presented scenario depends on the number of tokens supplied to P0, produced by P2 and passed to P3, as well as the number supplied to P4 and P5 via P8.

6.2.1 Feasibility Analysis SDF Transformation

As the logical level of the SeCo Service description framework may contain many access patterns with overlapping input and output attribute domains, an analysis of reachability within the selected set of access patterns is necessary in order to select the combination of access patterns with the full reach.

The scenario outlined in Figure 6.2 depicts the situation where access patterns share input and output attribute domains, thereby facilitating data surfacing through an input/output conduit of interconnected pattern materializations. The access pattern *TheaterByPhone* utilizes the *reseeding input strategy*, as the method of populating its input attribute values dictionary. The *Theater.Phone* input attribute is obtained from the equivalent output domain attribute's *Theater.Phone* values during the materialization process. Access Pattern **TheaterByCity** depends on external supply of values for the *Theater.City* input attribute as is the case for **MovieByTitle** access pattern being dependant on the external supply of values for *Movie.Year, Movie.Title*

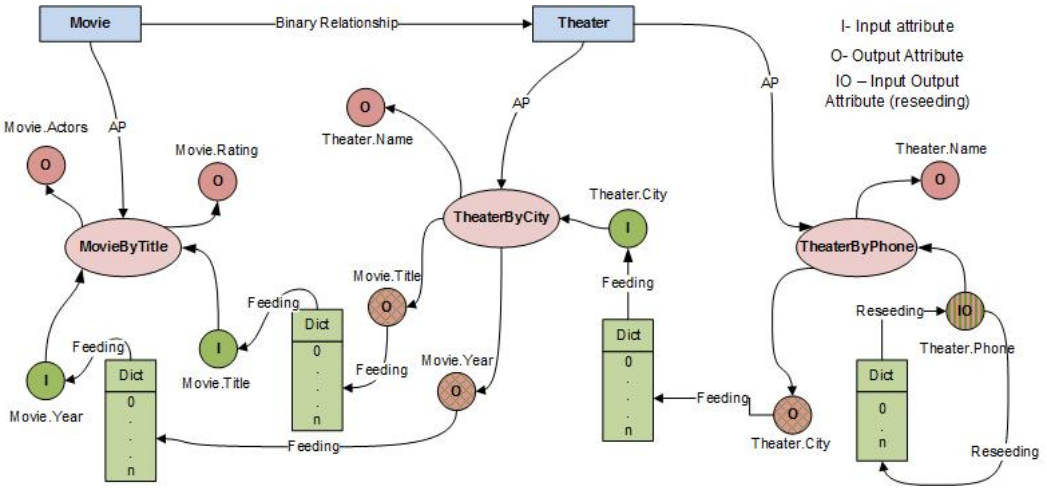


Figure 6.2: Example of schema of the materialization problem at AP level and its Petri Net representation.

input attributes. As illustrated in the Figure 6.2, the output attribute values *Theater.City* of the **TheaterByPhone** access pattern are passed - to *TheaterByCity* input attribute Theater.City in order to maintain the materialization process of the latter. In turn, the *output attribute* values Movie.Title and Movie.Year of **TheaterByCity** are passed to the input attributes belonging to the same domain Move.Year, Movie.Title of the **MovieByTitle** access pattern in order to perform its materialization.

The above access pattern interaction can be further represented as a bipartite graph where the input attribute domains form a set of vertices and the output attribute domains form another set of vertices of the graph. In the context of the MPMS scenario, we model the directed bipartite graph as a Petri net in which:

- The input and output attributes and their corresponding domains are a set of places,
- The input and output attribute domain values are represented as *tokens* that propagate through the network,
- The query is the transformation from input to output domain as well as the process of supplying output attribute domain values to the input

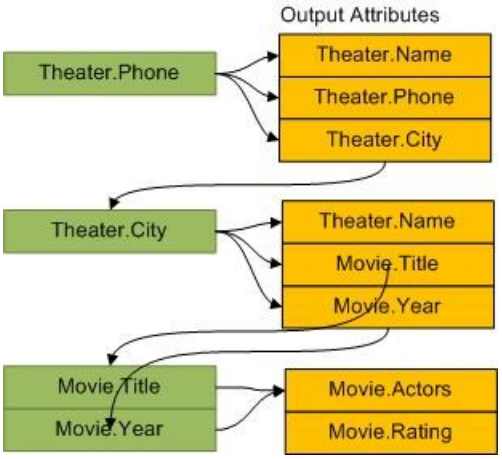


Figure 6.3: Bipartite graph representation of the access patterns.

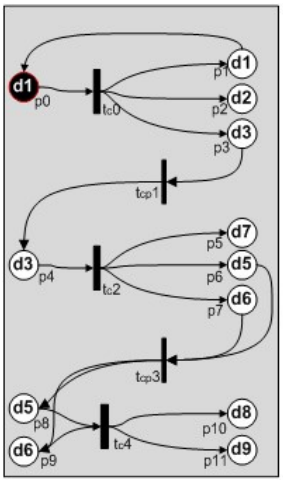


Figure 6.4: Petri Net representation of the MPMS example scenario with token in p0.

attribute domain via an available connection pattern,

- The places and transitions constitute the nodes of the graph connected via directed arcs. Places may contain zero or more tokens, labelled with input and output attribute domain values and their types,
- The state often referred to as *marking* M, is the distribution of tokens over places. In the MPMS scenario the state is the distribution of values over input and output attribute domains of the involved access pattern.

Formally, an access pattern presented as a Petri Net ap_{pn} is a triplet (P_{ap}, T_{ap}, F_{ap}):

- P_{ap} is a finite set of places,
- T_{ap} is a finite set of transitions ($P \bigcap T = \varnothing$),
- $F_{ap} \subseteq (P_{ap} \times T_{ap}) \bigcup (T_{ap} \times P_{ap})$ is a set of arcs (flow relation).

As followed in Chapter 4 we extend the latter definitions:

- $P_{ap} = \{I,O,R_a\}$ is a set that consists of three finite subset of input I, output O, and ranking R_a attributes of ap,
  - $I \in P$, where I is the set of attributes $a_1 ... a_j \subseteq A$ that defines the input interface of ap,

- $O \in P$, where O is the set of attributes $a_1...a_k \subseteq A$ that defines the output interface of ap,
- $R_a \in P$ where R_a is the set of attributes $a_1...a_p \subseteq O$ that defines the ranking condition of ap.

- $T_{ap} = \{t_c, T_{cp}\}$ is a set that consist of a single transition t_c and set T_{cp} where:
  - $t_c \in T$ where t_c is a transition that facilitates a materialization call $c \in C$, as defined in chapter 4,
  - $t_{cp} \in T_{cp}$ where t_{cp} is a transition that facilitates execution of the connect pattern cp,
  - cp is defined as tuple $< a_I, a_O >$ where attribute $a_I \in I$, an input interface of $ap_{pn,1}$ and attribute $a_O \in O$, an output interface of $ap_{pn,2}$.

A place p is called an *input place* of a transition t if and only if there exists a *directed arc* from p to t. In the MPMS scenario an input place is typically an *input domain attribute* of an involved access pattern **or** an *output domain attribute* of an access pattern connected by a connection pattern to another access pattern's input domain attribute. A place p is called an output place of transition t if and only if there exists a directed arc from t to p. In the MPMS scenario an output place is an output domain attribute of the associated access pattern.

Tokens, which are input and output attribute domain values propagate through the net, driven by firing of the queries. Transitions are active components in a Petri net. In the MPMS scenario, there are two semantically different transitions: *materialization call transition* **and** *connection pattern transition* (output domain to input domain supply transition). Transitions change the state of the net according to the following *firing rule*:

- A transition t is said to be enabled if and only if each input place p of t contains at least one token - a value d in either input or output domain of the associated access pattern,

- An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and *produces* one token for each output place p of t. In the MPMS scenario, the actual transition firing represents either materialization or supply of values (tokens) from output attributes,

- In this way a sequence of states $M_0, M_1, ...M_n$ is generated, such that M_0 is the initial state and M_{i+1} is the state reachable from M_i by firing a transition. If several transitions are enabled at the same time, then any of these transitions may be the next to fire. In the MPMS scenario this means that as soon as there are input domain attribute values available to the access pattern, a query q_k^p is issued and the produced result's r_k^p values are placed in the output domain attributes of the involved access pattern. Alternatively, if there is a value in the output domain attribute connected by connection pattern to another access pattern's input attribute domain the supply transition is fired and the value passed over,

- Let M_0 be the initial state of a Petri net. In the MPMS scenario the initial place is input attribute(s) of at least one involved access pattern,

- A state is called a *reachable state* if and only if there exists a firing sequence $M_0, M_1, ...M_n$ which enables this state. In the MPMS scenario this sequence is a combination of both places whose arcs contain a *materialization call transition* and places whose arcs contain connection pattern transition,

- A terminal state is a state where none of the transitions is enabled, i.e. a state without successors. In the MPMS scenario the terminal state is either a place representing an output domain attribute with no outward connections or a place representing an output domain attribute for which a materialization call did not produce any values valid in the associated input attribute domain.

We evaluate an MPMS topology as a marked net $< PN, M_0 >$ where $< PN, M0 >$ with $PN = (P, T, F, D, M_0)$ specifies:

- An initial marking (i.e., state) M_0 - an *initial input attribute* belonging to access pattern for which there are satisfied input attribute domain requirements,
- The rules of evolution T, F - a set of *directed* transformations between input to output domains in the same access pattern and between access patterns in form of *connection patterns*,
- A set of places P in input and output domains,
- A set of tokens (values) D that propagate through the net.

The reachability graph of MPMS topology $< PN, M_0 >$ is a firing sequence $G = (X, E, \sigma, x_0)$ where:

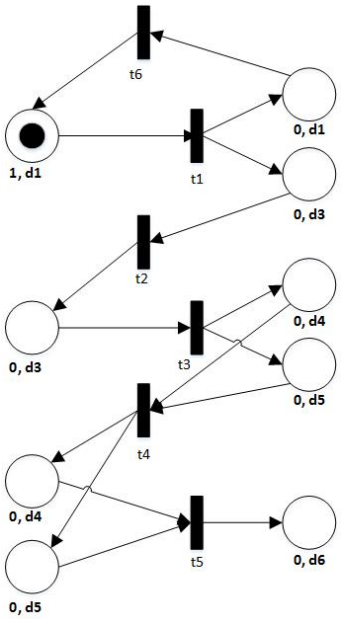
- $X = R(PN, M_0)$, i.e., the states of the sequence are the reachable markings,
- $E = T$, i.e., the events in MPMS execution are the transitions of the net,
- For any two reachable markings M, M':
  - A reachable markings transition $\sigma(M, t) = M' \Leftrightarrow M[t > M'$, i.e., there exists an arc labelled t from M to M' on the sequence if and only if marking M' is reachable from M on firing transition t;
- $x_0 = M_0$, i.e., the initial state of the sequence is the initial marking.

In the example below we demonstrate all the reachable (or feasible) states in series of firing sequences for the given MPMS topology.

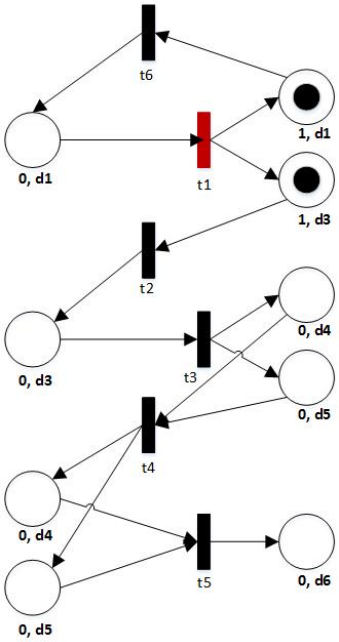
We present MPMS topology with three access patterns
$AP1$ with I=$\{d1\}$, O=$\{$ $d1$, $d3\}$, $AP2$ with I=$\{d4$, $d5\}$, O=$\{d6\}$ and $AP3$ with I=$\{d3\}$, O=$\{d4$, $d5\}$. The logic of the firing sequence is defined by the following reachability algorithm.

Given an MPMS net $< PN, M0 >$ let G be its reachability graph with set of states X constructed using the defined algorithm.

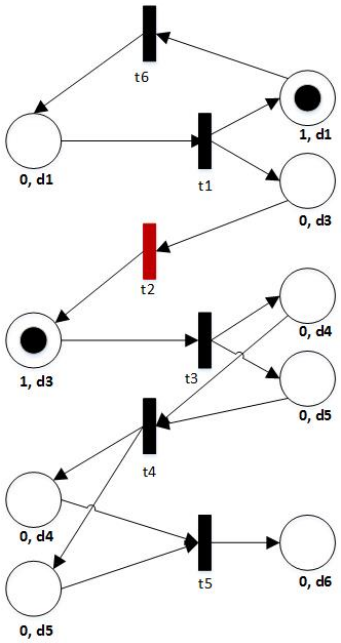
- $R(N, M_0) = X$ - the reachability set,



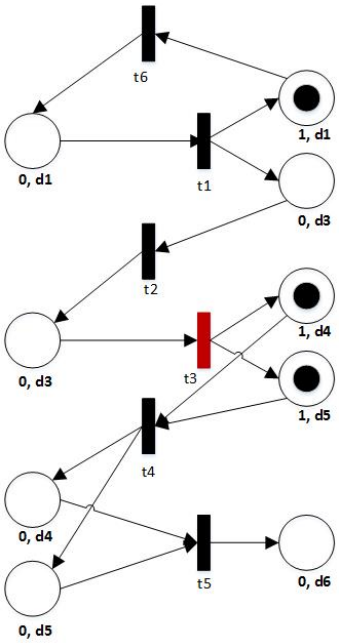
Initial Marking M0 = [1,0,0,0,0,0,0,0,0,0]



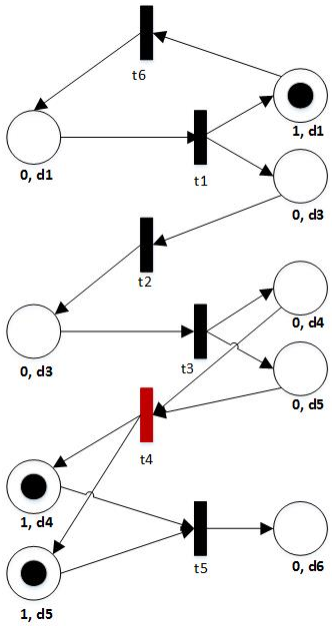
Upon t1 firing M1 = [0,1,1,0,0,0,0,0,0,0], Transitions t2 and t6 are enabled



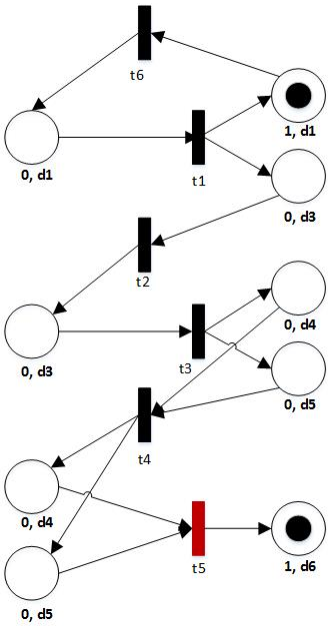
Upon t2 firing M2 = [0,1,0,1,0,0,0,0,0,0], transition t3 is enabled



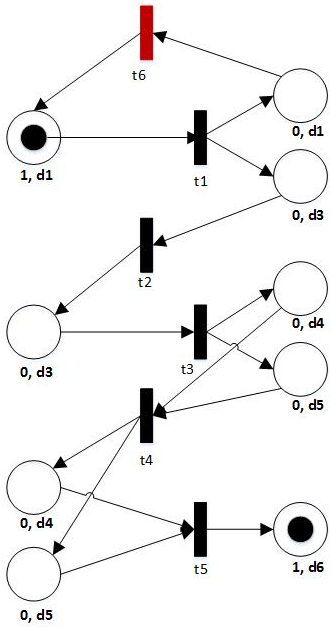
Upon t3 firing M3 = [0,1,0,0,1,1,0,0,0,0], transition t4 is enabled



Upon t4 firing M4 = [0,1,0,0,0,0,1,1,0], transition t5 is enabled



Upon t5 firing M5 =[0,1,0,0,0,0,0,0,1]



Last, upon t6 firing M6 =[1,0,0,0,0,0,0,0,1]

Algorithm 5 Algorithm for reachability Graph

```
1: Consider as root node the initial marking M_0 for which there is an AP with I = ∅ and tag it new.
2: while a node tagged new exists do
3:     Select a node M tagged new
4:     for all t enabled at M do
5:         Let M' be the marking reached from M by firing t
6:         if there does not exists a node M' in the graph then
7:             add a new node M' and tag it new
8:         end if
9:         Add an arc labelled t from M to M'
10:    end for
11:    Untag node M
12: end while
```

- $L(N, M_0) = L(G)$ - set of all possible firing sequences from M_0.

We retrieved the following information from the reachability graph G.

- Marking M is reachable $\leftrightarrow$ M is a node of G,
- $\sigma \in L(N, M_0) \leftrightarrow \delta(M_0, \sigma)$ is defined in G,
- $M[\sigma > M' \leftrightarrow M[t > M'$ there exists a path as arc t from M to M' labelled by σ.

As follows in the example sequence PN[0,1,0,0,0,0,1,1,0] is reachable for: $M_0(d_0)$=$\{1,0,0,0,0,0,0,0,0\}$ $L(PN,d_0)$=$\{t1, t2, t3, t4, t5\}$ => $L(PN,d_0)$=$\{0, 1, 0, 0, 0, 0, 0, 0, 1\}$.

6.3 Feasibility Analysis Formalization

6.3.1 Single Access Pattern

We further expand the definition of access pattern Petri Net in subsection 6.2.1 to be exploited within the single pattern single service (spss) materialization scenario. In the spss context we define a Petri Net pn_{ap} that models access pattern ap, as a tuple $< P_{ap}, T_{Cap}, F_{ap}, D_{ap}, M_{0,ap} >$ where:

- $P_{ap} = \{P_I, P_O\}$ is a set of 2 finite subsets of places where:
    - Input Places subset $P_I = \{p_1, p_2, ..., p_n\}, p_n \in I$, where I is the set of attributes $a_1...a_j \in A$ that defines the input interface of ap,

- A place $p \in P_{ap}$ belongs to P_I subset if and only if there exists a directed arc from p to $T_{C,ap}$.

- $P_O = \{p_1, p_2, ..., p_m\}, p_m \in O$, where O is the set of attributes $a_1 ... a_k \in A$ that defines the output interface of ap,
  - A place $p \in P_{ap}$ belongs to P_O subset if and only if there exists a directed arc from $T_{C,ap}$ to p.
- $T_{c,ap}$ is a transition that facilitates materialization call $c \in C$,
- $F_{ap} \subseteq (P \times T) \bigcup (T \times P)$ is a set of arcs (flow relation),
- $D_{ap} = \{D_I, D_O\}$ is a set of 2 subsets of tokens where:
  - $D_I = \{d_1, d_2, ..., d_n\}, d_n \in dV_I$ where dV_I is a superset of all input value sets $dV_1, ...dV_j$ mapped to the attributes in input domain I,
  - $D_O = \{d_1, d_2, ..., d_m\}, d_m \in dV_O$ where dV_O is a superset of all output value sets $dV_1 ... dV_j$ mapped to the attributes in output domain O.
- $M_{0,ap} : P_I \rightarrow \{0, 1, 2, 3, ..., k\}$ is the initial marking, where k is the number of places in P_I,
- $P \bigcap T = 0$ and $P \bigcup T \neq \varnothing$.

The dynamic behaviour of a Petri Net pn_{ap} is described as a change of state according to the following transition - *firing rules*:

1. A transition $T_{c,ap}$ is said to be enabled if each input place p_n of $T_{c,ap}$ is marked with one token $d_n \in D_I$,

2. A firing of an enabled transition $T_{c,ap}$:
   - Removes tokens $d_n \in D_I$ from each input place p_n of $T_{c,ap}$, and adds tokens $d_m \in D_O$ to each output place p_m of $T_{c,ap}$,
   - If output place p_m is a duplicate of input place p_n of $T_{c,ap}$ that is if both places map semantically and by type identical attributes $a_j, a_k \in A$, then the token $d_m \in D_O$ is propagated to input place p_n of $T_{c,ap}$ as well - as in Figure 6.5.

A **reachability** property of a Petri Net pn_{ap} is defined as a sequence of firing of transitions which would result in transforming a marking $M_{0,ap}$ to $M_{i,ap}$, where $M_{i,ap}$ represents the specific state in which all places in P_O are populated by tokens, and the sequence of firings represents the required functional behaviour, that is a *firing rule* as defined above. Such state is illustrated in Figure 6.4 if firing of transition place t_{c0} resulted in valid values at places *p1, p2* and *p3* of the top most access pattern (TheaterByPhone).

The set of all possible markings reachable for initial marking M_0 of a Petri Net pn_{ap} is called the reachability set and is denoted by $R_{ap}(M_{0,ap})$. The set of all possible firing sequences from $M_{0,ap}$ is denoted by $L_{ap}(M_{0,ap})$.

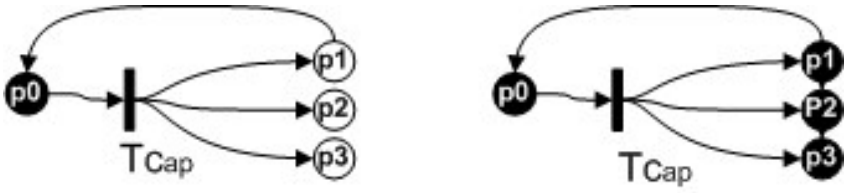


Figure 6.5: Petri Net pn_{ap} reachable firing sequence with input output domain connection.

Initial marking $M_{0,ap} = \{1, 0, 0, 0\}$ reaches sequence $M_{i,ap} = \{1, 1, 1, 1\}$ following sequence of transition firings - $T_{c,ap}$. $R_{ap}(M_{0,ap}) = \{p0, p1, p2, p3\}$, while $L_{ap}(M_{0,ap}) = \{T_{c,ap}\}$ - as illustrated in Figure 6.5.

Or initial marking $M_{0,ap} = \{1, 0, 0, \}$ reaches sequence $M_{i,ap} = \{0, 1, 1\}$

Figure 6.6: Petri Net pn_{ap} reachable firing sequence.

following sequence of transition firings - $T_{c,ap}$. $R_{ap}(M_{0,ap}) = \{p1, p2\}$, while $L_{ap}(M_{0,ap}) = \{T_{c,ap}\}$ - as illustrated in Figure 6.6.

An **unboundedness** property of a pn_{ap} is derived from the definition of *boundedness* where a Petri net is said to be k-bounded if the number of tokens in any place p, where $p \in P_{ap}$, is always less or equal to k (k is non-negative integer number) for every marking M reachable from initial marking

$M_0, M \in R_{ap}(M_0)$. Thus, a Petri net pn_{ap} is said to be unbounded if there is a place $p \in P_{ap}$ that can hold an arbitrarily large number of tokens, as in Petri net example in Figure 6.7 where places *p0* and *p1* are unbounded.

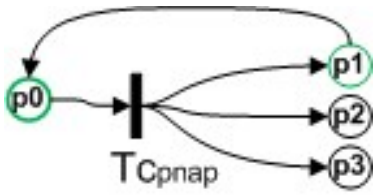


Figure 6.7: Petri net that is unbounded.

6.3.2 Multiple Access Patterns

We further expand the definition of access pattern Petri net in subsection 6.2.1 to be exploited within the multi pattern multi service (mpms) materialization scenario. In the mpms context we define a Petri Net pn_{mp} that models a combination of multiple access patterns mp, as a tuple $< PN_{mp}, CP_{mp}, T_{c,mp}, P_{mp}, D_{mp}, M_{0,mp} >$, where:

- $PN_{mp} = \{pn_{ap1}, pn_{ap2}, ..., pn_{apn}\}$ is a set of all Petri nets that model access patterns contained in mp net,
- $CP_{mp} \subseteq CP$, where CP is a set of connection patterns, and CP_{mp} is a subset of all connection patterns connecting access patterns in mp net,
- $T_{c,mp} = \{t_{c,mp1}, t_{c,mp2}, ..., t_{c,mpn}\}$ is a finite set of transitions that facilitate the interactions between $pn_{apn} \in PN_{mp}$ as enabled by connection patterns in CP_{mp},
- $D_{mp} = \{D_{ap1}, D_{ap2}, ..., D_{apn}\}$ is a set of tokens that includes token sets of all access patterns in mp net,
- $P_{mp} = \{P_{ap1}, P_{ap2}, ..., P_{apn}\}$ is a superset of all place sets of all access patterns in mp net,
- $M_{0,mp} : P_{mp} \rightarrow \{0, 1, 2, 3, ..., k\}$ is the initial marking in the domains of mp, where k is the number of places in P_{mp},

- $P \bigcap T = 0$ and $P \bigcup T \neq \varnothing$.

The dynamic behaviour of a Petri Net pn_{mp} is described as a change of state according to the following transition - *firing rules*:

1. A transition Tc_{mp} is said to be enabled if place $p \in P_{apn}$ that matches $a_{ap,I}ofcp_{mp} \in CP_{mp}$ is marked with one token $d_{ap} \in D_{apn}$,

2. A firing of an enabled transition T_{cmp} copies tokens $d_{ap} \in D_{apn}$ from each output place $pofP_{apn} \in P_{mp}$ that matches $a_{ap,I}ofcp_{mp} \in CP_{mp}$, to each input place $p \in P_{apn}$ that matches $a_{ap,O}ofcp_{mp}$.

A reachability property of a Petri Net pn_{mp} is defined as in section 6.3.1 and expanded to all Petri Nets pn_{ap} that participate in pn_{mp}.

The set of all possible markings reachable for initial marking M_0 of a Petri Net pn_{mp} is called the reachability set and is denoted by $R_{mp}(M_{0,mp})$. The set of all possible firing sequences from $M_{0,mp}$ is denoted by $L_{mp}(M_{0,mp})$. The set of all access pattern Petri nets from $M_{0,mp}$ is denoted $P_{mp}(M_{0,mp})$. Figure 6.8 depicts a reachable solution where the initial marking $M_{0,mp} = \{1,0,0,0,0,0,0\}$ reaches sequence $M_{i,mp} = \{1,1,1,0,1,1,1\}$ following sequence of transition firings - $T_{cap1}, t_{cmp1}, T_{cap2}$. $R_{mp}(M_{0,mp}) = \{p0, p1, p2, p3, p4, p5, p6\}$, while firing sequence $L_{ap}(M_{0,mp}) = \{T_{cap1}, t_{cmp1}, T_{cap2}\}$ and the set of access patterns is $P_{mp}(M_{0,mp}) = \{pn_{ap1}, pn_{ap2}\}$.

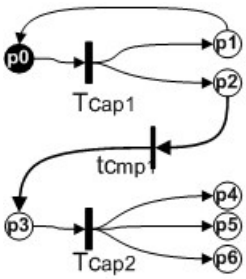
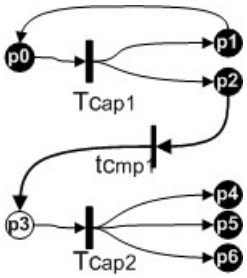


Figure 6.8: MPMS Net reachable solution.



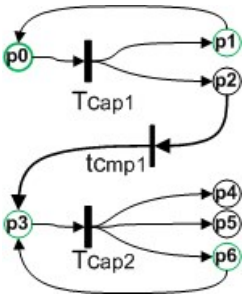


Figure 6.9: Unbounded MPMS Net Places.

An **unboundedness** property of a Petri Net pn_{mp} is defined as in section 6.3.1. Thus, a Petri net pn_{mp} is said to be *unbounded* if there is a place

$p \in P_{mp}$ that can hold an arbitrarily large number of tokens. In the Petri net example in Figure 6.9 places *p0, p1* and p3, p6 are unbounded.

6.4 Feasibility Analysis Model

In the following subsections we will study the reachability of an MPMS network by looking into three factors: Initial Access Pattern Choice, Transformations Direction of the connection patterns and Connection Patterns between Access patterns in the examined MPMS topology.

6.4.1 Initial access pattern

Firstly, we investigate effect of the choice of the *initial access pattern* i.e., the placement of the token as the initial marking M_0 to the reachability of the proposed MPMS topology. To illustrate the given problem let us assume, tak-

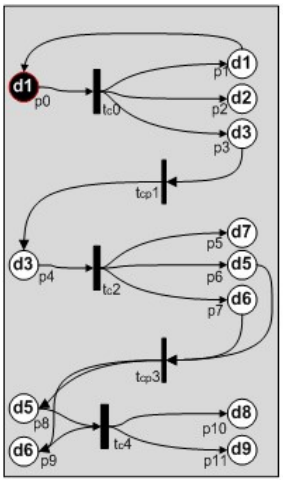


Figure 6.10: TheaterByPhone - initial AP.

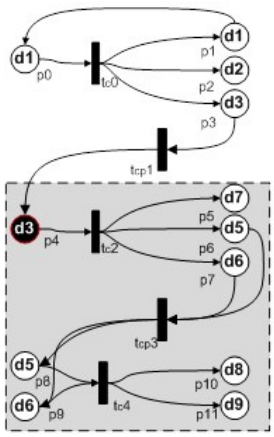


Figure 6.11: TheaterByCity - initial AP.

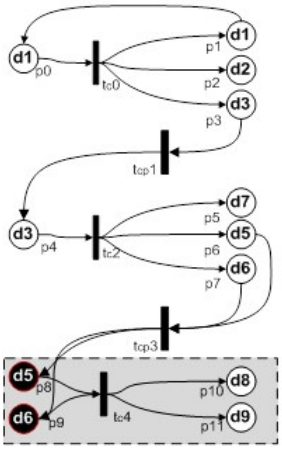


Figure 6.12: MovieByTitle - initial AP.

ing the example in Figures 6.10 that the *TheaterByPhone* access pattern is chosen as the initial access pattern with an appropriate starting dictionary of input values and the token placed to $P0$ is M_0. The access pattern *TheaterByCity* is dependant on the output tokens produced by the materialization of the *TheaterByPhone* and the number of queries that is able to execute is

limited by the number of non-duplicate output values tokens that the preceding access pattern supplies to it via the materialization call transition t_c1. Equivalently, the third access pattern *MovieByTitle* executes as many queries as many output tokens it receives from the *TheaterByCity* access pattern via connection pattern transition $t_{cp}3$.

As a comparison, if *TheaterByCity* access pattern was chosen as the initial access pattern - the token placed to $P4$ is M_0 as in Figure 6.11, the number of queries executed would depend solely on the size of the input dictionary, as the access pattern layout is not supporting the reseeding input strategy. Consequently output tokens supplied to the depending *MovieByTitle* access pattern via connection pattern transition $t_{cp}3$ would be dependent on the size of the *TheaterByCity* input dictionary and the number of non-duplicate output values (i.e., tokens) this materialization produces. Most importantly, due to the topology of the graph, the *TheaterByPhone* access pattern would not execute at all, producing null materialization output.

In the most limited representation of the topology, if *MovieByTitle* is the initial access pattern - the token placed to $P8$ and $P9$ as M_0 as in Figure 6.12, the only access pattern reached is *MovieByTitle*, producing as many queries as the initial dictionary supplies input tokens, while both *TheaterByPhone* and *TheaterByCity* are unreachable i.e., there is no materialization produced for these two access patterns.

The choice of the initial access pattern place in the given topology affects the total size of the materialization produced. In the case of *TheaterByPhone* if the token is placed to $P0$ as state M_0, being the initial pattern, the self-driven materialization of all three access patterns is limited by reseeding volume of the place $P1$ - $d1$ output domain attribute - as illustrated in Figure 6.10. A partial materialization of dependent access patterns is limited by the size of their input attribute domain dictionaries, i.e., the number of tokens placed to *P4, P8* and *P9*, and the non-duplicate output attribute values produced in the form of tokens by *P6* and *P7*, as illustrated in Figures 6.11 and 6.12.

6.4.2 Transformations Direction

A further factor affecting the reach of the materialization is observed through the *directions of the connection pattern transformations* in the selected MPMS topology. In order to further understand the situations affecting the reachability of the access pattern connection pattern dependencies let us consider the following change of the *MovieByTitle* access pattern with addition of the new connection pattern, as shown in Figure 6.13. In the output attribute domain of the *MovieByTitle* we added *Theater.City* (*d3*) attribute also present in the input attribute domain of *TheaterByCity*, thus forming new connection pattern between these two access patterns via transition $t_{cp}5$.

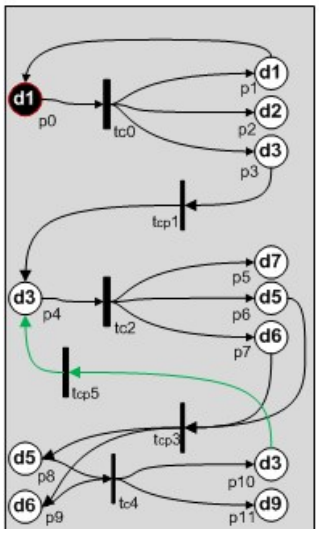


Figure 6.13: Petri Net model of the topology with added connection pattern.

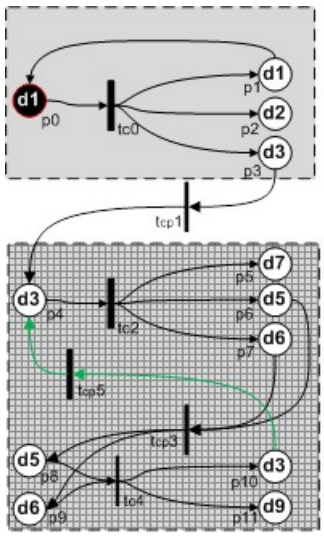


Figure 6.14: Two sub-topologies derived on the basis of the new connection pattern.

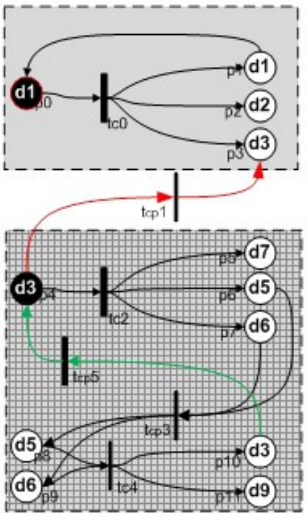


Figure 6.15: Reversal of the connection pattern flow between sub-topologies.

As shown in Figure 6.13, the newly added connection pattern transition $t_{cp}5$ between *MovieByTitle* and *TheaterByCity* removes dependencies between the *TheaterByPhone* access pattern and the rest of the network. Nevertheless, the choice of the initial access pattern is still important as the choice of *TheaterByPhone* as the initial pattern provides the largest reach by initiating *MovieByTitle* and *TheaterByCity*. It also facilitates the connection pattern between *MovieByTitle* and *TheaterByCity* via transition $t_{cp}5$, thus

making them independent, in case *d3* output domain attribute value supply from *TheaterByPhone* exhausts tokens from place *P3*.

Furthermore, the change forms two self-driven materialization sub-topologies. Figure 6.14, illustrates the sub-topologies and the importance of *TheaterByPhone* and *TheaterByCity* in the d3 arc direction (sequence P3, t1, P4) as the choice of either *MovieByTitle* or *TheaterByCity*. The initial access pattern prohibits *TheaterByPhone* from materializing. Clearly, it is necessary to choose *TheaterByPhone* and either *MovieByTitle* or *TheaterByCity*. Thus tokens can be placed to $P0$ and $P4$ or $P0$ and $P8$, $P9$ as state M_0 to reach all parts of the network.

As an addition to the example in Figure 6.14, it becomes evident that the materialization of this layout may be observed as two separate materializations; the top one being driven by the SPSS concept and the bottom one by the MPMS concept. The top one is used to reinforce the $d3$ input attribute value supply via connection pattern transition $t_{cp}1$ in case the supply from connection pattern transition $t_{cp}5$ is unavailable or degraded by duplicate values, as illustrated in Chapter 7. Figure 6.15 shows how the change of the arc direction between *TheaterByCity* and *TheaterByPhone* via transition $t_{cp}1$ disconnects *MovieByTitle* and *TheaterByPhone* sub-topology from the rest of the network, thus making it impossible to reach each other.

6.4.3 Connection patterns between access patterns

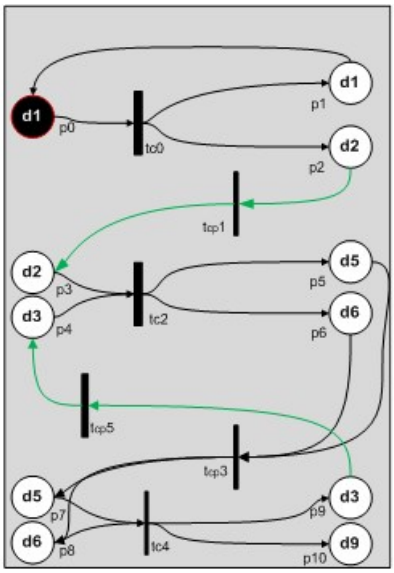


Figure 6.16: More than one connection pattern per access pattern.

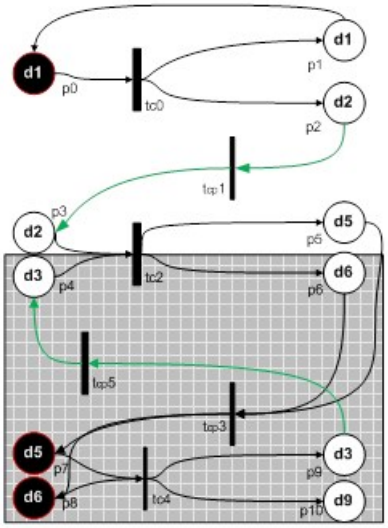


Figure 6.17: Two initial access patterns as a seed.

Figure 6.16 shows the topology, whereby an access pattern shares connections with more than one access pattern via connection pattern transitions $t_{cp}1$ and $t_{cp}5$. Here, it is necessary to choose more than one access pattern as a starting seed of the MPMS topology. As demonstrated in Figure 6.17, the only way to reach all parts of the graph is to start both the top and the bottom access patterns via places $P0$, $P7$, $P8$ as state M_0.

As demonstrated, the choice of the initial access patterns, the direction of the connection pattern arcs and the number of connection patterns per AP play a crucial role in reachability of the whole network.

6.4.4 Proposed Feasibility Analysis Model

In this section we will derive a set of formal rules to provide a basis for establishing the topology with the largest reach.
As demonstrated in the previous section, the reachability of MPMS topology (MPMSNet) is affected by three factors: Initial Access Pattern Choice, Arc Direction of the connection patterns and Number of connection patterns per AP in the examined MPMS topology..

Strongly connected and boundedness [Murata 1989] properties of the Petri Net are used to model reachability analysis of the MPMS materialization scenario. By using these properties two method propositions for the analysis of the MPMS reachability are derived and proved.

We define a strongly connected component of MPMS Petri net topology as the subset of its places such that for every two places x and y in the component, there is a directed path leading from x to y.

Strongly connected components i.e., a firing sequence of places such that there is a set of arcs from each place in the sequence to every other place in the sequence can be:

- Transient components - there are paths going out of the component,
- Ergodic (or absorbing) components - there are no paths going out of the component.

Proposition 1 If a starting place M_0 is chosen from an access pattern representing a strongly connected subnet, whereby outgoing paths exist that lead from an output domain to an input domain, then all places contained within the MPMS can be reached.

Proof Places M_0 and M_1 in a given MPMS are connected if there is a path t_1 connecting these places. Likewise, places M_1 and M_2 are connected if there is a path t_2 connecting these places. In general, for any given n, there must be a path t_n connecting M_{n-1} to M_n since all places M_k, $k = 0, ..., n$ are chosen from a transient strongly connected subnet. Thus, for the chain, $M_0, M_1, ..., M_{n-1}, M_n$, it follows that there is either a path directly linking path M_i to M_j, for all $0 < i < j < n$, or a path exists as a series of traversals

M_i to M_{i+1}, M_{i+1} to $M_{i+2}, ..., M_{j-1}$, to M_j. In either case, given a starting place M_0 all other nodes are reachable from M_0 by traversing a certain number of links.

Given an MPMS net $<PN, M_0>$ let G be its reachability graph with set of states X constructed using the algorithm 5 in subsection 6.2.1.

As depicted in Figures 6.18 and 6.19: If $M_0(d1)$={1,0,0,0,0,0,0,0,0} then

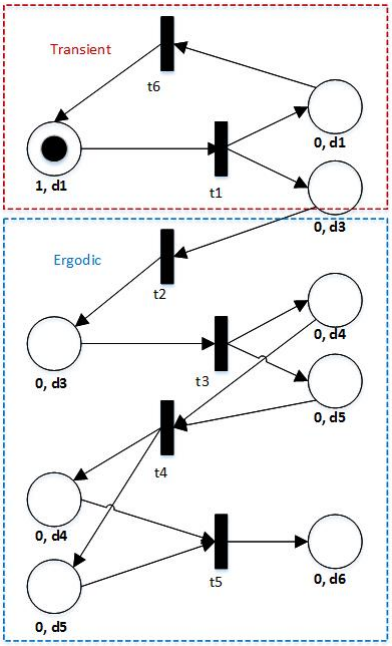


Figure 6.18: Transient Initial Place $M_0(d1)$

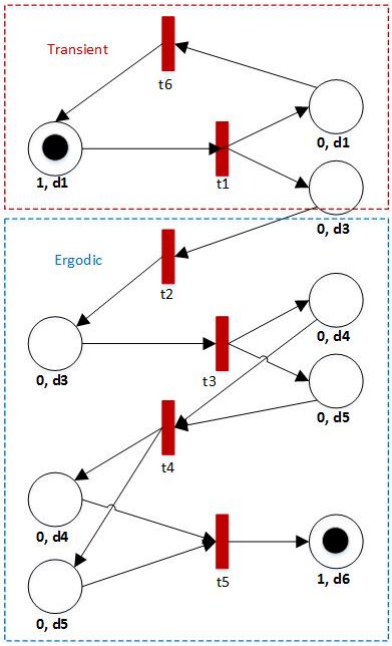


Figure 6.19: $M_0(d1)$ reachable position

PN={1,0,0,0,0,0,0,0,0,1} is reachable $L(PN, d1)$ ={t1, t2, t3, t4, t5, t6} => $R(PN, d1)$={1,0,0,0,0,0,0,0,0,1} = X - reachable.

Contradicting, as depicted in Figures 6.20 and 6.21: If $M_0(d3)$ = {0,0,0,1,0,0,0,0,0} than PN={1,0,0,0,0,0,0,0,1} is not reachable. $L(PN, d3)$={t3, t4, t5} => $R(PN, d3)$={0,0,0,0,0,0,0,0,1}
$L(PN, d1)$!= $L(PN, D3)$ and $R(PN, d3)$!= $R(PN, d1)$ - not reachable.

To further illustrate, let us consider examples in subsections 6.4.1, 6.4.2 and 6.4.3. As depicted in Figures 6.10 and 6.11 a strongly connected transient component is formed by *TheaterByPhone* access pattern driven by the reseeding data surfacing process via input and output attribute **Theater.Phone (d1) - places P0 and P1**. TheaterByPhone forms a strongly connected component through **d1** connection; further, it connects (transient

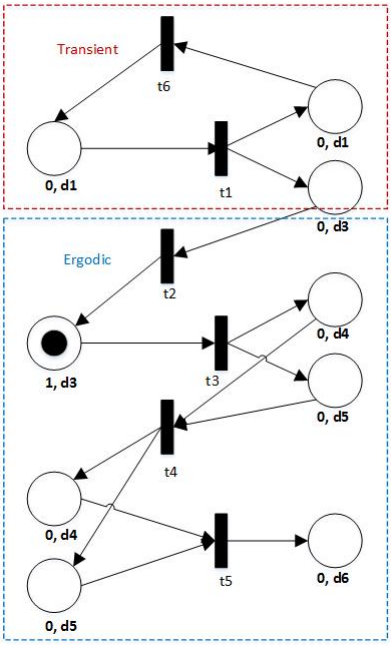


Figure 6.20: Ergodic Initial Place $M_{0,e}$.

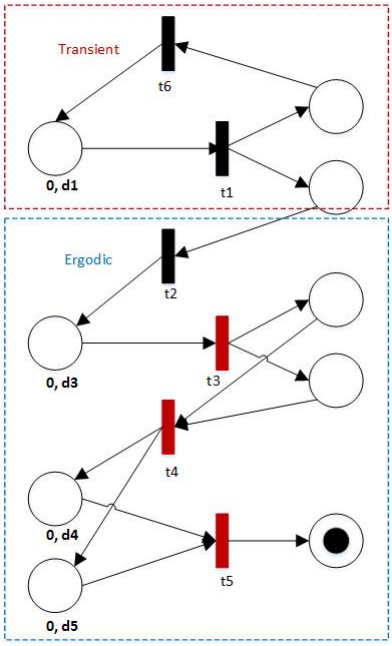


Figure 6.21: $M_0(d3)$ reachable position.

property) via **d3** to the other sub-topology. By reversing the outward arc i.e., by making it ergodic, the rest of the topology is disconnected thus, making it unreachable - Figure 6.15.

Figures 6.13, 6.14 and 6.15 introduce another strongly connected component to the topology. The new component is composed of *TheaterByCity* and *MovieByTitle* access patterns. This component is larger than the first strongly connected component (TheaterByPhone), and as demonstrated in Figure 6.15, once this component's outward arc is directed towards the smaller strongly connected component via transition t_{cp1}, the topology connects the sub-topologies but it does not reach all the places yet.

As shown in Figures 6.16 and 6.17, a topology where an access pattern shares input connections with more than one access pattern Proposition1 holds, in order to satisfy access patterns with more than one input connection pattern, it is necessary to choose initial access patterns in both sub-topologies of the graph. Hence, as demonstrated in Figure 6.16 the only way to reach all parts of the graph is to start both at the top and bottom access patterns; that is, to define places *P0, P7* and *P0* as state M_0.

As defined in section 6.3.1 an MPMS Petri net $< PN, M >$ is bounded if

and only if, for every reachable state and every place p the number of tokens in p is bounded.

Proposition 2 The Boundedness of given subgraph G (strongly connected component) is determined by the highest bound of the attribute (place) in the output domain that makes a transient relationship with any subgraph's input domain that connects to G.

Proof For graph G we select all places with transient relationship as set S $\{M_1,...,M_n\}$ that produce values in their respective domains. We now choose an element from S that produces the maximum number of values k from its domain. This determines that the bound for graph G is now k as no other places in S are able to produce more values.

For the marked net PN $<d1, d3, d1>$ reachability graph $R(PN,d1)$

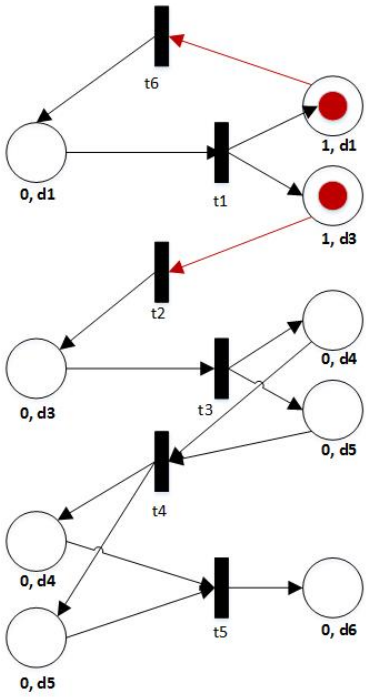


Figure 6.22: Boundedness proposition example.

$L(PN,d1)$={t1, t6} => [1,0,1] the bound k of place p is max $M(p)$ for all places in R - Figure 6.22. The max $M(p)$ is defined by the filling of $d1$ output attribute domain.

If output $k1 = |M(d1)| = 3$, $L(PN,d1) = \{t1, t6\}$ executes 3 times.

If output $k2 = |M(d1)| = 2$, $L(PN,d1) = \{t1, t6\}$ executes 2 times.
As $k1 > k2$ the bound k_p of place p is max $M(p)$ for all places in R.

For the marked net $PN <d3, d4, d5, d4, d5, d6>$ reachability graph $R(PN,d3)$ $L(PN,d3)$={t2,t3,t4,t5} => [0,0,0,0,0,0,1] the bound k of place p is max $M(p)$ for all places in R - Figure 6.22. The max $M(p)$ is defined by the filling of $d3$ output attribute domain.
If output $k1 = |M(d3)| = 4$, $L(PN,d3) = \{t2t3t4t5\}$ executes 4 times.
If output $k2 = |M(d3)| = 2$, $L(PN,d3) = \{t2t3t4t5\}$ executes 2 times.
As $k1>k2$ the bound k_p of place p is max $M(p)$ for all nodes in R.

To further illustrate, let us consider the topology in Figure 6.14. The bound of the top subgraph - TheaterByPhone - is defined by the bound of **d1** output domain attribute, place **P1** (transient place **tp1**), since this attribute holds a transient relationship with the input domain. In other words as many non-duplicate values that it delivers to the input domain, that many queries will be executed.

The bottom subgraph's bound is determined by the highest of the bounds of *d3, d5 or d6*, i.e. places *P6, P7, P10* as they all equally contribute to the input domain of the subgraph.

Once the reachability graph as defined in section 6.3.1 and determined via Proposition 1 is established, Proposition 2 is used to determine how many times this layout can be reached i.e., the maximum number of materialization calls that can be executed in the given MPMS net.

6.4.5 Algorithms

Algorithms 6 and 7 present the feasibility analysis split into two steps according to the propositions defined in section 6.4.4. Step1 algorithm analyses

Algorithm 6 Feasibility Analysis Step 1

```
1: N_CP // number of connection patterns
2: AP_MPMS // set of ap in MPMS
3: ap // access pattern
4: scp // strongly connected component
5: O− > I // output to input
6: for each ap in AP_MPMS with N_CP > 0 do
7:     for each scp linking ap with O− > I outgoing arcs do
8:         Start ap in scp
9:     end for
10: end for
```

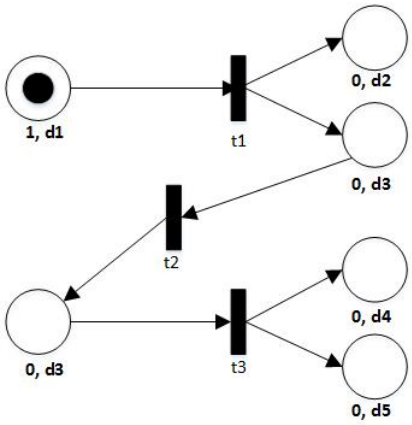


Figure 6.23: Step 1 Base case start position.

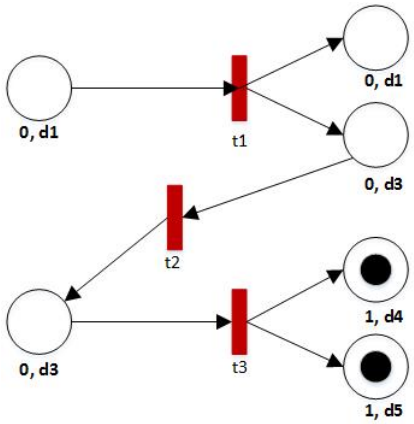


Figure 6.24: Step 1 Base case reachable position.

the given MPMS net for a strongly connected component in order to start the execution of an initial AP so that the largest reach of the given topology is ensured. Step 2 further analyses the model by establishing the Boundedness of the given MPMS net.

As a base case, as depicted in Figures 6.23 and 6.24, we consider PN_{MPMS} = {ap_1<d1,d3,d1>,ap_2<d3,d4,d5>} , ap_1 contains $N_{cp} = 1$, ap_1 is also a *scp* with $O \rightarrow I$ as it is connected to ap_2 via cp<$d3_{ap1,O}$,$d3_{ap2,I}$>.

We choose $M(d1)$ as M_0 of the reachable solution. If $M_0(d1)$ = {1,0,0,0,0,0} is reachable

$L(PN, d1)$={t1, t2, t3} => $[0, 0, 0, 0, 1, 1]$ thus $R(PN, D1)$ is reachable.

As a more elaborate example as depicted in Figures 6.25 and 6.26 we con-

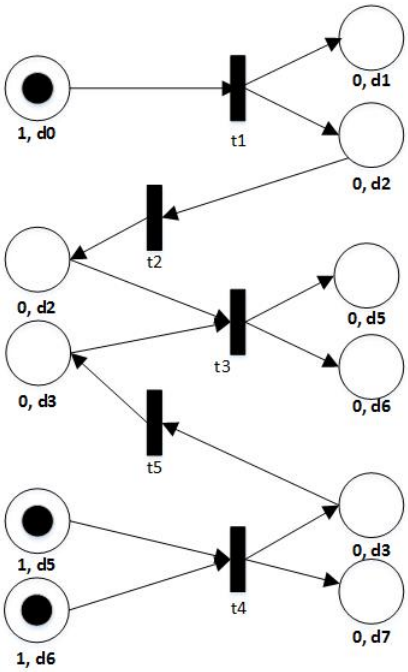


Figure 6.25: Step 1 Complex case start position.

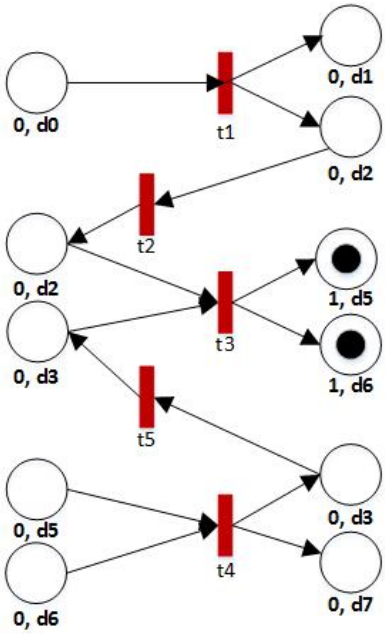


Figure 6.26: Step 1 Complex case reachable position.

sider PN_{MPMS} = {ap_1<d0,d1,d2>,ap_2<d2,d3,d5,d6>, ap_3<d5,d6,d3,d7>}.
Access pattern ap_1 contains $N_{cp} = 1$, ap_1 is also a *scp* with $O \rightarrow I$ as it is connected to ap_2 via cp<$d2_{ap1,O}, d2_{ap2,I}$>. We choose $M(d1)$ as M_0 of the first *scp* of reachable solution. Access pattern ap_3 contains $N_{cp} = 1$, ap_3 is also an *scp* with $O \rightarrow I$ as it is connected to ap_2 via cp<$d3_{ap3,O}, d3_{ap2,I}$>.
We choose $M(d5, d6)$ as M_0 of the second *scp* of reachable solution.
If $M_0(d0, d5, d6)$ = [1,0,0,0,0,0,1,1,0,0] is reachable.
$L(PN_{MPMS}, (d0, d5, d6))$ = { t1 t4 t2 t5 t3} => [0,0,0,0,0,1,1,0,0,0,0] thus $R(PN_{MPMS}, (d0, d5, d6))$ - reachable.
Complexity: $O(n2)$ for worst case scenario.

Step 2 further analyses the model by establishing the Boundedness of the subgraphs of the given MPMS net.
As a base case - Figure 6.27 - we consider PN_{MPMS} {ap_1<d1,d2,d1>}, this net forms one *scp* with $d1$ from output making connection with $d1$ in the input

Algorithm 7 Feasibility Analysis Step 2

```
1: I // input domain
2: tp // transient place
3: scp // strongly connected component
4: for each scp in MPMSNet < PN, M_0 > with tp arcs directed to I do
5:     if List < tp > .count > 1 then
6:         scp bound = Max(M(tp)) for all nodes in List < tp >
7:     else
8:         scp bound = M(tp)
9:     end if
10: end for
```

domain. Thus bound of PN_{MPMS} $Max(P2)$.

Complex example - Figure 6.28 - features PN_{MPMS} {ap_1<d1,d2,d1>, ap_2<d3,d1,d5>}, there are two *scp*; top one having two bound places with $Max(P2)$ or $Max(P5)$ determining the *scp* bound.

Complexity: $O(n2)$ for worst case scenario.

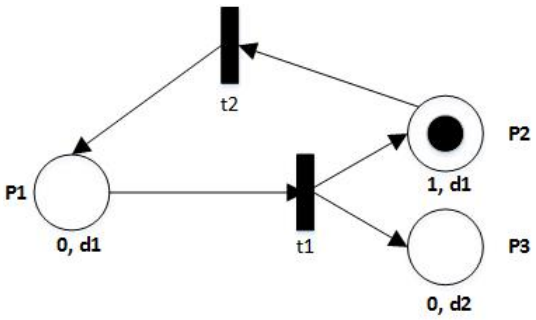


Figure 6.27: Step 2 Base case.

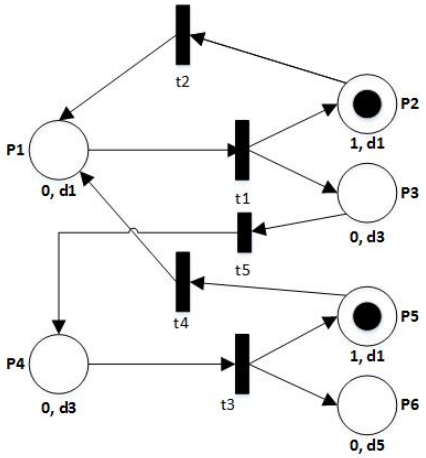


Figure 6.28: Step 2 Complex case.

6.5 Empirical Study

In this section we demonstrate the performance of our approach in the context of the feasible solutions delivered by the proposed model algorithms. The performance of the proposed model was evaluated by the number of executed queries (firing sequences) against MPMS layouts with different number of bound places and strongly connected components.

6.5.1 Experimental Settings

To evaluate the proposed solution we apply the model to a set of materialization tasks designed to support a 'reallocation search' multi domain search scenario depicted in Figure 6.29. It is illustrated by real life queries such as 'I am looking for a Java developer position in an area where housing is in a certain price range and there are good Italian and Thai restaurants nearby', against a service repository which deals with a variety of domains from jobs to real estate to entertainment and general shopping. Each task is constrained by a predefined set of input dictionaries and required materialization coverage. The feasibility analysis is performed against a set of potential access patterns $|AP| = 120$, derived from the Domain Diagram. The empirical study considered the access patterns 1) *jobByLocation* (AP1) which takes job position location and job type as input attributes - Figure 6.29 in green colour - $I = \{JobKeyword, Location\}$ and - Figure 6.29 in red colour $O = \{JobName, SalaryRange, JobType, Location\}$; 2) *realEstateByLocation* (AP2) which takes a real estate type, price range and location as input attributes - Figure 6.29 in green colour - $I = \{realEstateType, PriceRange, Location\}$ and - Figure 6.29 in red colour $O = \{PriceRange, NoOfBedrooms, Location\}$; 3) restaurantByLocation (AP3) which takes a cuisine type and location as inputs - Figure 6.29 in green colour $I = \{cuisineType, Location\}$ and - Figure 6.29 in red colour $O = \{Name, Rating, Cuisine, Location\}$. For practicality reasons all solutions recommended by the feasibility analysis were restricted to MPMSNet with strongly connected component size of $2 <= |SCP_{MPMs}| <= 5$, and the number of transient places $1 < |tp| <= 10$. For each AP at least one feasible solution was chosen for each combination of feasibility properties; for each feasible solution 10 runs were performed; the materialization goal was set to 50 executed queries/retrieved result sets. The input dictionaries for attributes that were not populated by reseeding or a connection pattern were supplied by static dictionary of 50 values belonging to their respective domains e.g., $dict_{cuisineType} = \{'Italian','Indian', ...\}$ or $dict_{realEstateType} = \{'house','flat','studio', ...\}$. To avoid bias, the input dic-

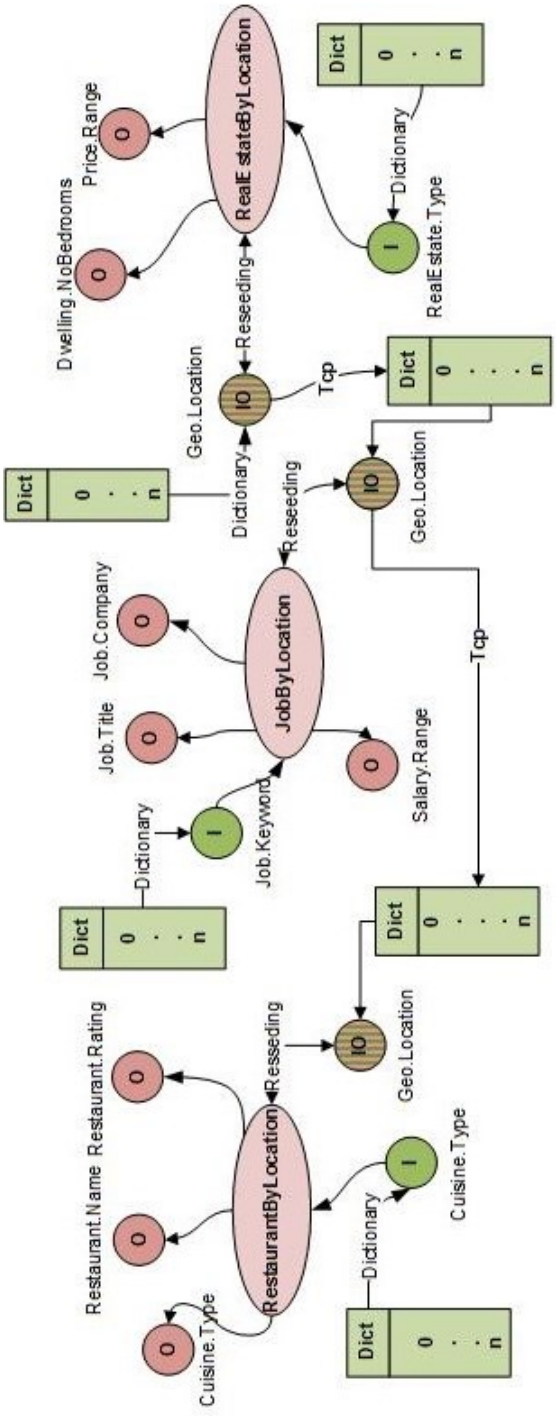


Figure 6.29: 'Reallocation search' multi domain search scenario (I-input, O=output, IO-input/output reseeding)

tionaries were initialized at each run by a randomly selected subset. To perform the evaluation, we created a master database composed of $\tilde{1}$00K listed items from several existing on-line real estate, restaurant rating and job sites.

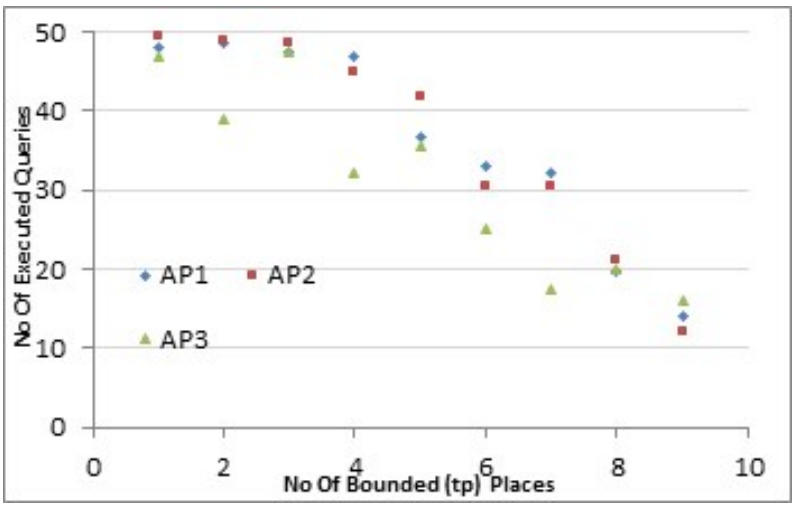


Figure 6.30: No of tp vs No Of Executed Queries.

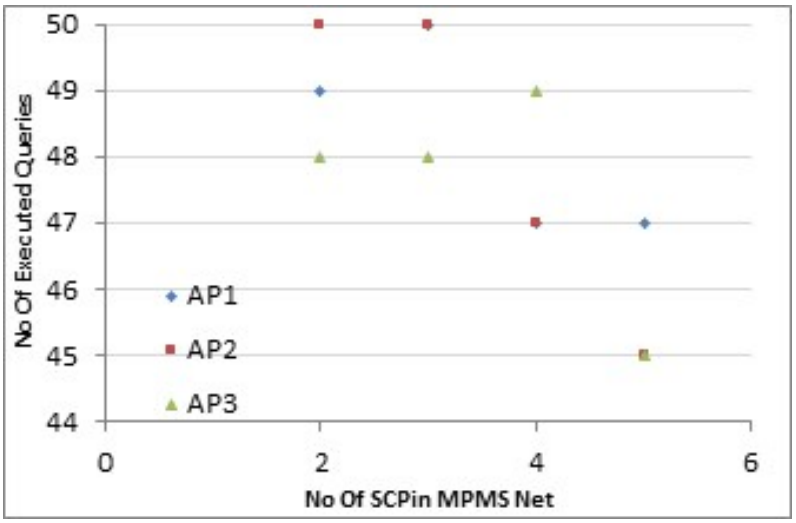


Figure 6.31: No of scp vs No Of Executed Queries.

Influence of transient places to the materialization. Figure 6.30 examines the effect of the number of transient (bound) places to the number

of executed queries (sequences). The result reveals an interesting trend as the number of queries decrease with an increase in the number of bound places. The layouts with less than 5 transient places seem to perform adequately, on the whole reaching the given goal of 50 queries. The materialization output (number of queries) decreases with increase in the number of transient places. This is most likely imposed by the unpredictable nature of the seeding output attributes 'capacity' as it is hard to predict number of valid input values produced by the transient place due to the unknown data distribution of the remote source.

Influence of strongly connected places to the materialization. Figure 6.31 examines the effect of the number of strongly connected components (scp) of the given feasible layout to the obtained materialization. The results are consistent in all 3 APs, as there is no significant shift in the obtained materialization volume as the number of *scp* increases. This further validates the importance of positioning of input values, as there was no deviation in the input dictionary sizes the executions performed close to 100% in each case.

6.5.2 Discussion

We consider real-life application of the empirical results in the context of Search Computing. Search Computing queries typically address search tasks that go beyond a single interaction, the task transpires via a query paradigm that supports multi-step, exploratory search over multiple Web data sources. This query paradigm requires users to be aware of searching over "interconnected objects" with given semantics, but each exploration step is simplified as much as possible, by presenting to users at each step simple interfaces, offering some choices that can be supported by the system; choices include moving "forward", by adding new objects to the search, or "backward", by excluding some objects from the search; and the selection and de-selection of displayed results in order to dynamically manipulate the result set.

SeCo service description framework's (SDF) Domain Diagram provides users a mean of expressing queries directly upon the concepts that are known to the system, such as jobs or restaurants, or salaries; moreover, users are

aware of the connections between the concepts, and they can, therefore, e.g., select a job and then relate it to several other concepts such as salaries, location or job type; or select a show and then relate it to several other concepts: the performing artist, the close-by restaurants, the transportation and parking facilities, other shows being played on the same night in town, and so on. The query is focused (and restricted) to known semantic domains provided via the linked ontology.

At any stage, users can "move forward" in the exploration, by adding a new object to the query, starting from the connections available in the SDF and from the objects that have been previously extracted following connections provided in Domain Diagram. Users can also "move backward" (backtrack) in the exploration, by excluding one of the objects from the query, or by "unchecking" some of their previous manual selections of relevant object instances. For example, a user may decide that the bus ride is too inconvenient, preferring to use a car instead, and then explore parking opportunities for the selected restaurants.

In such a dynamic search environment our solution provides the search with a means of constructing a quick materialization 'pathway'. The feasibly analysis considers access patterns linked to the concepts in the given Domain Diagram then constructs a feasible solution and enables materializer to quickly satisfy new search need.

This is particularly applicable in a situation where at any stage of the exploration, users can store the status of their search, by saving the query that has been formulated so far as well as the results. As the dynamic nature of the exploration does not always guarantee data validity, as obviously relevant data may not be available, quick feasibility analysis and small goal materialization effort further reinforces the available, stored results. Therefore, the same query can be repeated for a returning or different user, saving the exploration effort. Potential expansion of the materialization by, for instance, including top k+1 results in the materialized data corpus may prove beneficial for backtracking at the level of individual conditions as well, as e.g., in changing the choice of job search keyword from "JAVA Developer" to "Web Developer" during reuse.

6.6 Chapter Summary

In this chapter we designed a method for analysing a potential section of access patterns in the given service description framework in order to determine a feasible combination of AP for which the given materialization task succeeds. Furthermore, we introduced and characterized feasible solution properties reachability and boundedness which were the building blocks of the proposed analysis. We have also presented a two-step algorithm based on the proposed analysis procedure. We performed a range of live materialization tasks using the presented algorithms and confirmed validity of the analytic step in the materialization formulation. We also came to an interesting observation in regards to the bound AP attribute in respect of the final materialization outcome. The experiment results have shown that our approach provides better performance in terms of final materialization coverage where the number of bound places is smaller in equally reachable AP layouts.

In the next chapter we will extend deeper into the materialization formulation by addressing the actual materialization run-time query and service selection optimization, relevant metrics and the cost model.

Chapter 7

Materialization Optimization

7.1 Introduction

Proposed materialization scenario may result in a reachability graph that involves access patterns with several available service interfaces. This event is reflected in SPMS and MPMS materialization scenarios illustrated in Chapter 4. In materialization run-time each of these services once queried may return tuples varying in duplicate saturation, result size or response time. The services may also be affected by inflexible service level agreement.

The number of queries and result set size may be compromised if:
a) the number of queries k is limited by input dictionary size, i.e., the available input value dictionary size is not sufficient to provide large enough rewrite,
b) provided input dictionary does not match a segment of the output domain contained in the source, i.e., thus expressed queries do not yield any results, e.g., queried source holds all movies in year range 1980 -1990 while the input dictionary contains values from year 2000 onwards,
c) effective result *pageSize* is decreased by the presence of duplicate tuples, i.e., some of the retrieved result tuples are already present in the materialized data,
d) *pageSize* is limited by the service itself as a part of their service level agreement.

Further, the materialization is compromised in terms of the effective processing time if: a) web source is characterized by *low response time*,
b) number of source invocations (queries) k is limited by service level agreement imposed by the remote data provider.
These factors affect the volume and the currency of the data harvested during materialization process, thus, compromising the accuracy of the dependent

multi domain search application.

To prevent these factors' detrimental effect on the materialization, it is necessary to further differentiate between services belonging to the same access pattern on the basis of *service interfaces properties* and *characteristics of the materialized data source*.

This differentiation is necessary in order to optimize materialization runtime.

The goal of optimization is to *minimize* the running time of the materialization process, while *maximizing* the produced materialization output RM and maintaining the right leverage between the performance and the quality of the obtained materialization. Here, we aim to achieve the least possible running time in the context of participating services and available input dictionaries for the prescribed materialization task.

In this chapter we outline and define our optimization approach in respect to the materialization process. Further, we look at the optimization variables and potential metrics across *service interface properties, materialized data characteristics* and properties derived from the solution feasibility analysis.

We propose a cost model that employs the proposed set of optimization metrics in order to provide quantitative differentiation between the involved services and provide a basis of the consequent run-time optimization algorithm. Lastly, we describe the set of algorithms implementing proposed optimization and prove their effectiveness in the empirical study.

7.2 Optimization approach

Our optimization approach is focused on the *data surfacing* phase of the materialization process. We intend to improve both parts of the *call2service coupling algorithm* - Figure 7.1.

First, in the query queue sequence de-queuing strategy we intend to deliver improvement to the existing query selection algorithm. The goal is to improve rate of discovery of the input domain values in the mapped output domain so as to minimize problem cases outlined below:

(i) Provided dictionary of input values is saturated with non-domain values,

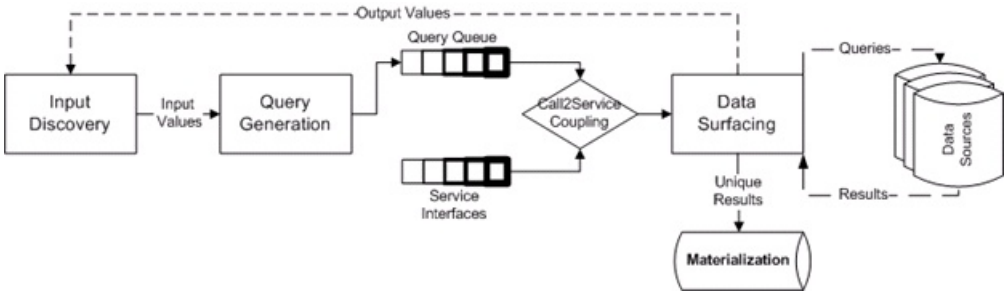


Figure 7.1: Materialization Process after optimization.

and
(ii) Input values build queries with low result expressive power, that is the obtained result sets are either empty or of small size.

Second, in the service interface selection strategy we intend to apply a new algorithm based on service execution cost per tuple. The goal of the algorithm is to minimize selection of services featuring:
(i) High duplicate levels,
(ii) Low $QueryResponseTime$ materialization property,
(iii) $CallsPerDay$ service level agreement (SLA) defined limits.

This implies transformation from set of available services SI to sequence of services $\{SI\}_{(i=1)}^{k}$ where $k = |SI|$ and it is ordered by the associated service interface execution cost.

We expect an improved query expressiveness and decreased duplicates rate to result in an improved materialization efficiency E_C, as fewer queries will be required to achieve desired coverage level.

Equivalently, we expect the selection of faster responding services with less SLA limit to improve materialization efficiency E_T as desired materialization coverage will be achieved over shorter period of time.

7.3 Optimization Strategy

The goal of materialization optimization is to *minimize* the running time of the materialization process for the given materialization scenario, while *maximizing* the produced materialization output RM and maintaining the right

balance between performance and the quality of the obtained materialization. Here, we aim to achieve the least possible running time in the context of participating services and available input dictionaries for the prescribed materialization task.

The optimization strategy goal is divided into two sub goals: a) Differentiation of services by performance; b) Differentiation of queries by expressiveness for the given source output domain.

7.3.1 Optimization Metrics Differentiation

In the subsequent subsection we illustrate influence of particular service and materialization properties by investigating how they affect the actual materializations performed in case study of Chapter 5. The materializations are labelled as **Mat1** and **Mat2** and present the data obtained during real-life materialization of the web data test bed as described in the case study. Data sets **Mat1** and **Mat2** represent materialized web source in the Real estate web domain, Mat1 featuring 200K records and Mat2 featuring 350K records. During the materialization web sources allowed for maximum result size rs<100 tuples and result chunk (page) size of 10 tuples. The obtained results were ranked (ordered) by geographical distance from the queried post code as reported in Chapter 5.

7.3.1.1 Service Interfaces properties differentiation

Service Interfaces wrap web sources that are in turn characterized by a set of dynamic, run-time properties related to their materialization performance - materialization properties and a set of static properties related to constraints imposed by the legal parameters as prescribed by the service provider. The run-time properties are used for **qualitative** differentiation between services. The qualitative analysis is based on performance cost per materialized tuple derived from the service interface materialization properties. A cost model applied in the service differentiation is based on a set of metrics derived from the service materialization properties.

We observe the effect of the service properties variables across all three

main classes of service properties as described in Chapter 4 section 4.2.1.1. We distinguish between the following properties relevant to the qualitative differentiation of the services.

Uniqueness Uniqueness properties are the group of service interface properties that indicate whether the service returns disjoint results for different inputs.

WithDuplicates Property of interest in the Uniqueness group of properties is *WithDuplicates* that defines whether the result sets r of two or more distinct queries q may contain the same tuples. Figure 7.2 illustrates the situation observed in real estate web source materializations described in case study of Chapter 5. In each materialization, there is a number of duplicate

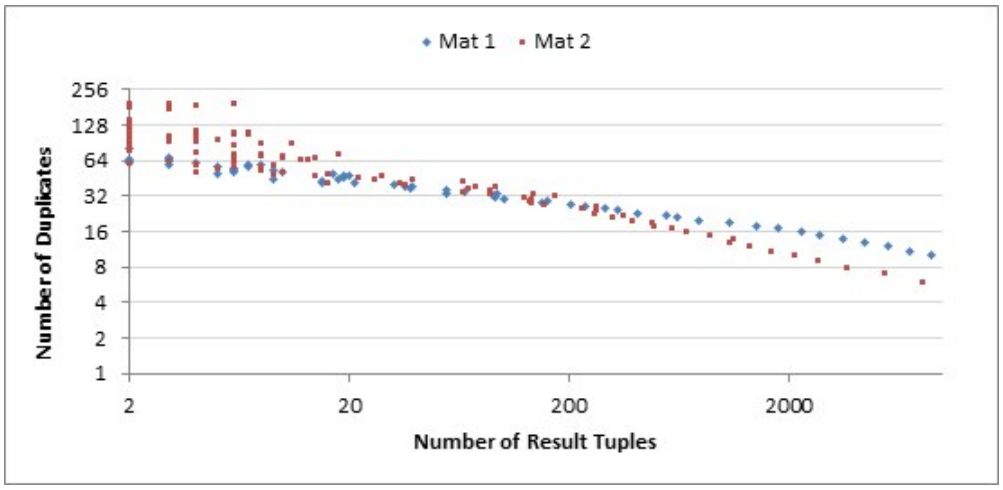


Figure 7.2: Distribution of duplicates over two performed materializations.

tuples discovered per query. *Mat2* materialization is characterized by high number of duplicates (over 100) for small set of tuples - in the range of 2 to 20. *Mat1* materialization expresses better utilization of the result set, as its maximum number of duplicates never exceeds 64 duplicates. Both of the materializations show steady duplicate discovery rate in the range 16 to 32 duplicates for roughly 2000 discovered tuples. Evidently, there was considerable waste of result set space per each issued query. Figures 7.3 and 7.4

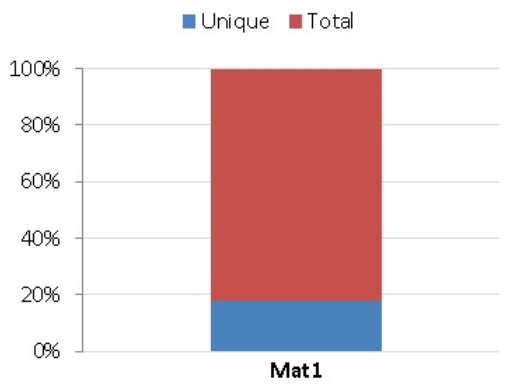


Figure 7.3: Total vs unique tuples in Mat1.

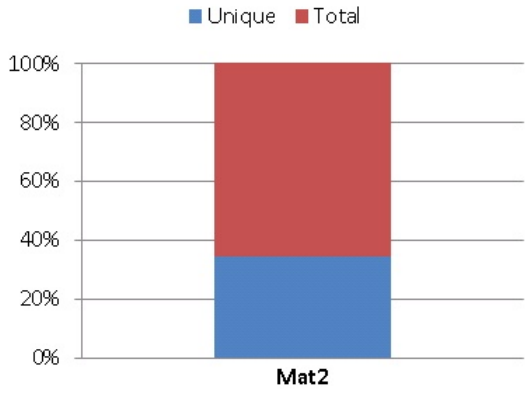


Figure 7.4: Total vs unique tuples in Mat2.

show the ratio of the total tuples versus the number of unique tuples in materializations $Mat1$ and $Mat2$. As the page size for $Mat1$ was 35 tuples per result the number of wasted queries were 39078; the page size for $Mat2$ was 100 with number of wasted queries 6013.

Following the above demonstration we define *WithDuplicates* metric, the uniqueness efficiency of the result r_p^k per each issued query q_p^k as a number of unique tuples discovered in the given result.

Definition 7.1. The materialization cost $cost[q_p^k, r_p^k]$ applied by the *WithDuplicates* property to the materialization R_M is defined as $cost[q_p^k, r_p^k] = size(r_p^k)/noUniqueTuples$. Both $size(r_p^k)$ and $noUniqueTuples$ are values related to the cardinality of r_p^k and are defined by the page size $1 \leq p \leq MaxPageSize$ where $1 \leq size(r_p^k) \leq MaxPageSize$ and $0 \leq noUniqueTuples \leq size(r_p^k)$.

For instance, if the discovered result set size is 10 and number of unique tuples is 10 then the materialization call cost per each tuple is 1, if the number of unique tuples changes to 5 then the cost per each tuple changes to 2, further, if the number of unique tuples is 2, the cost increases to 5.

Performance Performance group of service properties describe the service in terms of performance as outlined in Chapter 4 section 4.2.1.1. We take interest in *maximum page size, maximum result set size* and *query response time* properties.

Maximum Page Size Figure 7.5 below shows the materialization $Mat1$ in terms of the issued number of queries and the returned result chunk size. As expected the result set page size has major effect on the number of issued queries per materialization, thus, having a significant impact on the efficiency of the materialization process. The size of the query set needed for the dis-

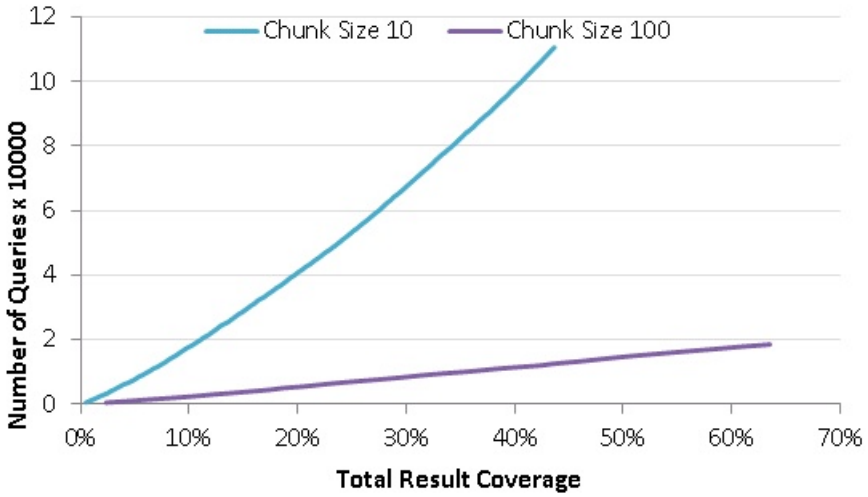


Figure 7.5: Total result coverage obtained with page size 10 and 100.

covery of 50% of the materialization corpus R_M is around 14000 queries for page size 100 while the query set size needed for the equivalent materialization with result set page size 10 - a 10-fold decrease - is close to 120000 queries, almost a 10-fold increase of the query set size as demonstrated in the actual materialization performed for the case study of Chapter 5 and outlined in Figure 7.5.

The experimental results clearly indicate reverse reciprocity between the result set chunk size and the number of queries needed for the materialization discovery, i.e., as the page size increases the number of queries decrease and vice versa - Figure 7.5.

Following the above demonstration we define *MaxPageSize* metric as the efficiency of the query q_p^k in terms of number of tuples in r_p^k discovered per each query.

Definition 7.2. The materialization cost $cost[q_p^k, r_p^k]$ applied by the

$MaxPageSize$ property to the materialization R_M is defined as: $cost[q_p^k, r_p^k] = MaxPageSize/C$, where $C <= MaxPageSize$, C is a constant, an arbitary value that determines the value range of this cost metric.

The cardinality of result set r_p^k is defined by the page size $1 \leq ps \leq MaxPageSize$ and by $1 \leq p \leq MaxNoChunks$ service properties.

For instance, if the prescribed maximum chunk size is 10 tuples per result set and constant is set C=100, the cost per tuple of this query is 0.1, if the chunk size is 50 the cost is 0.5 and if it is 100 tuples per result the cost of tuple is 1.0.

Maximum Result Size Following the same analogy as in Maximum Page Size it is fair to conclude that service interfaces with large *Maximum Result Size* property enable us to reach materialization corpus R_M with lesser number of materialization calls than if the maximum result size was mediocre. The latter scenario implies larger input value dictionary and an increased number of materialization calls, thus, rendering the materialization procedure inefficient.

Hence, we apply the same logic as in *MaxPageSize* metric to define the metric *MaxResultSize* as the efficiency of the query q_p^k in terms of number of tuples discovered per each result r_p^k.

Definition 7.3. The materialization cost $cost[q_p^k, r_p^k]$ applied by the $MaxResultSize$ property to the materialization R_M is defined as: $cost[q_p^k, r_p^k] = MaxResultSize/C$, where $C <= MaxResultSize$, C is a constant, an arbitary value that determines the value range of this cost metrics.

The cardinality of result set r_p^k is defined as for the *MaxPageSize* metric by the page size $1 \leq ps \leq MaxPageSize$ and by $1 \leq p \leq MaxNoChunks$ service properties.

Query Response Time Query Response time is the performance service interface property that measures end-to-end response time (e.g., average response time for each page). Figure 7.6 shows distribution of query response times per number of queries for materialization *Mat1*. Majority of the queries

are in the response time range 400ms - 3000ms, with maximum query time being close to 8000ms. Figure 7.7, shows a snapshot of the query response time

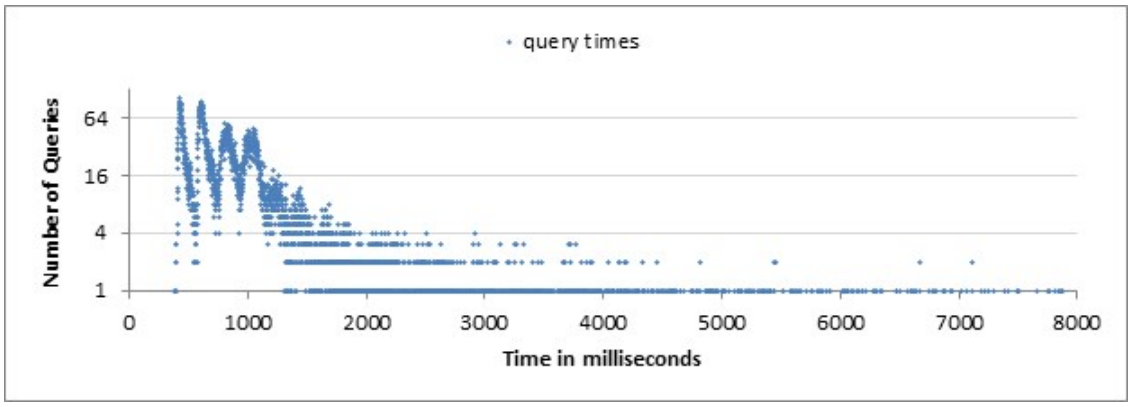


Figure 7.6: Distribution of query response times per number of queries in Mat1.

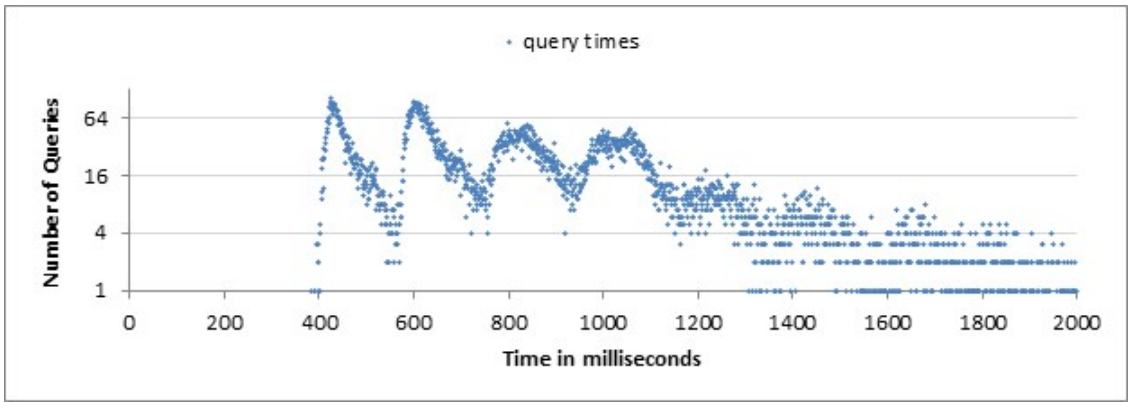


Figure 7.7: Snapshot of query response times per number of queries distribution.

distribution presented in Figure 7.6, here we focus on time range from 0ms to 2000ms. The figure clearly indicates that the fastest response time was around 400ms while the majority of the queries fit in the 400ms - 1500 ms range. The number of queries used was 25786, with total time of 25012210ms - 6.9 hours - giving an average query time of 969.9ms - for the whole materialization.

Thus, by utilizing the query response time property, we derive *ResponsePerQuery* metric as effective per-query response time of the given service.

Definition 7.4. The materialization cost $cost[q - p^k, r_p^k]$ applied by the *ResponsePerQuery* property to the materialization R_M is defined as:

$cost[q_p^k, r_p^k] = (1 - 1/t_{qi})$, where t_{qi} is response time of executed query q_p^k. Evidently, as the response time increases, the query cost increases asymptotically approaching 1 as the query time tends to infinity.

For instance, if the query time is 400ms the cost is calculated as (1 - 1/400) = 0.9975 or in the worst case of $Mat1$ materialization (1-1/8000) = 0.999875.

Service Level Agreement Properties Service Level Agreement (SLA) properties are the group of properties characterising services in terms of the level/quality of service offered by the data source service provider.

Daily invocation limit Daily invocation limit is SLA property limiting daily number of service invocations i.e., limiting the number of queries issued per day.

Typically if the service is constricted by an SLA agreement, the service user is required to sign the SLA agreement upon which the user is issued with an access key uniquely identifying the user with the service provider. On the basis of the supplied key as a part of the input signature the service provider tracks and limits the daily number of service invocations.

This scenario was in effect during the materialization $Mat2$, where the daily invocation limit was prescribed to 1000 calls per day. The materialization $Mat2$ was obtained by issuing 46000 queries over a time span of 46 days. The average query response time of the daily lots was less than 1000ms per query, thus, leading to conclusion that if there was not for the daily limit of calls the $Mat2$ materialization would be obtained within one day.

Clearly the SLA daily invocation limit brings the efficiency of the materialization process to a standstill. Hence, we define $CallsPerDay$ metric as materialization query efficiency in terms of SLA driven daily invocation limits.

Definition 7.5. The materialization cost $cost[q_p^k, r_p^k]$ applied by $CallsPerDay$ property to the materialization R_M is defined as: $cost[q_p^k, r_p^k] = DailyInvocationLimit/C$, where C is a constant value maintaining the ratio between *CallsPerDay* costs for different services. Constant C is defined by the smallest daily limit of the participating services.

For instance, if we consider three participating services with $DailyInvocationLimit$ limit of 1000 for si_1 , 2500 for si_2 and 100000 for si_3 the C would be assign limit of si_1 ie., $C = 1000$. Consequently the cost per query for si_1 = 1; si_2 = 2.5 and for si_3 = 0.01

7.3.1.2 Domain values quality differentiation

This set of characteristics is specific to each output domain used in the Reseeding input value discovery scenario. Each output domain logically connected to the input domain is characterized by the properties of the attribute values in the domain and how they appear in the materialization discovery.

For each $v \in O$ (output domain), domain value statistics are recomputed after each materialization call; in *query generation phase* the queries generated using the values are queued in descending order from largest to smallest according to *domain statistics scores*, and popped from the queue in *result generation phase* of the materialization process.

In order to present the matrices related to the attribute domain characteristics let us observe the following definition of the materialization RM.

A structured web source fronted by service interface and subject to the materialization process can be seen as a single relational table with n data tuples $t1, t2, ..., tn$ over O the set of attributes $a_1...a_k \in A$ that defines the output interface of ap.

As defined in Chapter 4, an attribute-value graph (AVG), G(V,E) for attribute's value domain dV_n is an undirected graph that can be constructed and formally defined as follows: for each distinct attribute value $d_i \in dV_n$ there exists a unique vertex $v_i \in V$. An undirected edge $e(v_i, v_j) \in E$ if and only if d_i and d_j coexist in one materialization call c_i where materialization call $c_i \in C$ is defined as a tuple $< q_p^k, r_p^k >$. Each edge e in AVG stands for a materialization call link between d_i and d_j.

According to the definition, attribute values from each materialization call may form a clique if two calls share the same attribute value. The corresponding vertex 'bridges' the two cliques.

Further, by characterizing structured web sources as AVG, materialization process is transformed into a graph traversal activity as described in Chapter

5. By starting with a set of seed vertices in the form of initial input dictionary, at each step, following the applied algorithm the materialization process selects a previously seen vertex v_i - a value in D, to discover all the neighbours of v_i.

Depending on the web source structure, an AVG is not necessarily fully connected. It may consist of several disconnected graphs that form an isolated cliques or data island in the total data corpus. Thus, an attribute value graph of an attribute in the output domain of service interface may form a disjoint union of graphs $U(V_1 \bigcup V_2, E_1 \bigcup E_2 \bigcup ... \bigcup V_{n-1} \bigcup V_n, E_{n-1} \bigcup E_n)$.

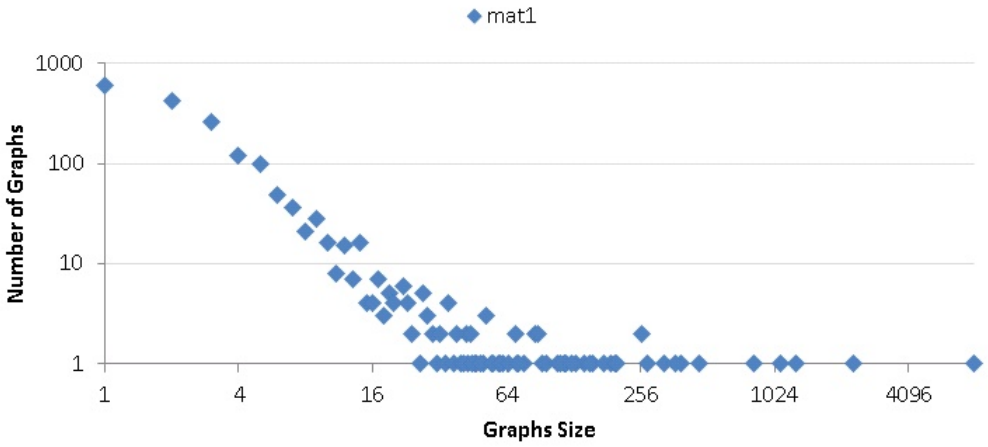


Figure 7.8: Disjoint graphs distribution in materialization Mat1.

Figure 7.8 shows the distribution of the disjoint graphs of Location output attribute domain, that has been logically connected to the equivalent attribute in the input attribute domain of the access pattern used in reseeding materialization scenario of $Mat1$ materialization process. There are a total of 1817 disjoint graphs - islands, largest graph containing 8093 vertices, and 605 graphs with only one vertex. Figure 7.9 shows the equivalent reseeding scenario as described in Figure 7.8, here applied to the materialization $Mat2$. Output attribute Location of the $Mat2$ service interface shows significantly less disjoint graphs - islands, than $Mat1$. $Mat2$ totals 35 disjoint graphs with the largest one having 42244 vertices out of total 43817 vertices in the location attribute domain - 90% of the total vertices. This implies significantly smaller

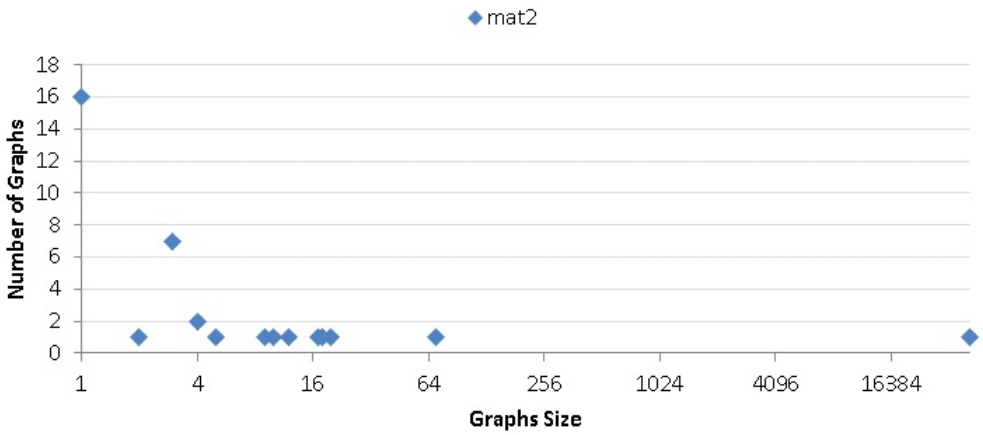


Figure 7.9: Disjoint graphs distribution in materialization Mat2.

attribute Location initial dictionary, as it is sufficient to start the materialization with 35 location attribute values, each one being an edge in one of the disjoint graphs to materialize the whole source.

In contrast, equivalent initial dictionary for $Mat1$ needs to contain 1817 location attribute values, each one being an edge in one of its 1817 disjoint graphs.

Evidently, this information becomes available once the actual materialization has been performed, thus, rendering this knowledge unavailable for the initial materialization. However, it may prove to be useful for the consequent materialization maintenance procedures where existing materialized data corpus is periodically refreshed following certain maintenance heuristics as hinted on in Chapter 8.

Following the above defined characteristics of the service interface attribute domains, we look into following attribute domain characteristics in order to define domain value optimization scores for values in domains used in Reseeding input discovery scenario. This optimization is to be exploited as part of general MPMS materialization optimization process and is described in the algorithm section further below.

Frequency Frequency property is derived by the intuition that the domain value having higher number of occurrences in discovered result set tuples has a

higher degree - the number of edges incident to the vertex - in the underlying graph representation of the given domain, thus leading to more undiscovered values - vertices.
We define *Frequency* as optimization metrics of reseeding attribute output domain domain value d_n in reseeding attribute output domain, as the number of occurrences of the value in each materialization call c result r_p^k until the value is used in query q_p^k - i.e., until it appears in the input domain.

Definition 7.6. The optimization score $score[d_n, dV_n]$ applied by *Frequency* metrics to the reseeding domain value d_n is defined as: $score[d_n, dV_n] = (1 - 1/f_{dn})$, where f_{dn} is the number of occurrences of the value d_n in each materialization call's result until the value is used in a query - i.e., until it appears in the input domain. The intention here is to assign higher scores to the values that appear more frequently in the output, thus, prioritizing such values in the reseeding process.

Example metric application: first, for all newly discovered and all previously discovered unused seeding values Frequency metrics are recomputed after each materialization call; second in *query generation* phase the queries generated using the values are queued in descending order from largest to the smallest according to *Frequency* metrics, and popped from the queue in *result generation* phase of the materialization process.

Harvest Rate Harvest Rate property observes number of result sets containing value d_n of the output attribute domain used in reseeding scenario.

We define *HarvestRate* metrics of reseeding attribute output domain value d_n in reseeding attribute output domain, as the count of results r_p^k containing the value d_n, until the value is used in query q_p^k - i.e., until it appears in the input domain.

Definition 7.7. The optimization score $score[d_n, dV_n]$ applied by *HarvestRate* metrics to the reseeding domain value d_n in dV_n is defined as: $score[d_n, dV_n] = (1 - 1/hr_{dn})$, where hr_{dn} is the count of result sets containing value d_n , until the value is used in query q_p^k.

As for the Frequency metrics the intention of the high score assignment is to reward and thus prioritize such values in the reseeding process. As opposed to low score earners that are penalized and used last. The same logic applies to all metrics and the derived scores in this group of properties.

Example metric application: first, for all newly discovered and all previously discovered unused seeding values *HarvestRate* metrics are recomputed after each materialization call; second in *query generation* phase the queries generated using the values are queued in descending order from largest to the smallest according to *HarvestRate* metric, and popped from the queue in *result generation* phase of the materialization process.

Number of Discovered Seeds The following metric is derived by the intuition that values coming for the result sets with larger diversification, i.e., higher degree of variation of the domain values, have better connectivity within their domain. We define *Diversification* metrics of reseeding attribute output domain value d_n in reseeding attribute output domain, as the count of new reseeding domain values discovered in result r_p^k and assigned to those values.

Definition 7.8. The optimization score $score[d_n, r_p^k]$ applied by *Diversification* metrics to the reseeding domain value d_n is defined as: $score[d_n, r_p^k] = 1 - 1/div_{r_p^k}$, where $div_{r_p^k}$ is the count of new reseeding domain values discovered in the result set r_p^k.

As in the metrics above the values with higher discovery rates are awarded and prioritized in reseeding.

Example metric application: for each result set a number of distinct newly discovered reseeding domain values is taken and associated with the values. Queries are arranged in the queue according to the *Diversification* metrics of the discovered values, the metrics and queue order are updated upon every materialization call execution.

Betweenness Centrality By graph theory definition, *betweenness centrality* is a measure of node's centrality in the network equal to a number of

shortest paths from all vertices to all others that pass through the node. In the context of reseeding attribute domain's attribute-value graph (AVG), this measure is a number of shortest paths that pass through value d_n.

Following the intuition that the high centrality of the value may lead to discovery of more unseen reseeding domain values we define *BetweennessCentrality* metrics as a number of shortest paths in domain of values dV_n that pass through the value d_n.

Definition 7.9. The optimization score $score[d_n, r_p^k]$ applied by *BetweennessCentrality* metrics to the reseeding domain value d_n is defined as: $score[d_n, r_p^k] = 1 - 1/bc_{dn}$, where bc_{dn} is a number of shortest paths in output domain dV_n that pass through the value d_n.

Example metric application: first for all newly discovered and all previously discovered unused seeding values the *BetweennessCentrality* metrics are recomputed after each materialization call; second in query generation phase the queries generated using the values are queued in descending order from largest to smallest according to *BetweennessCentrality* metric, and popped from the queue in result generation phase of the materialization process.

7.3.2 Optimization Cost Model

The primary objective of the cost model is to provide a measure of the *effectiveness* of the materialization process by computing the *ratio* between the performance *cost* incurred by service calls execution and the materialization *coverage* achieved during the materialization process.

Thanks to the formulation of a cost model, it is possible to optimize the materialization process by adjusting the execution variables (e.g., prioritize services or queries in the queue) that affect the performance of the process in order to increase the cost effectiveness of the materialization.

For the given materialization, the task differentiation between participating services is based on the ratio between the accumulated cost and the achieved materialization coverage per each service. Furthermore, a service that achieves more result coverage with less cost is prioritized on the service queue during the materialization process.

Concretely, in the materialization process context the objective of the cost model is to quantitatively characterize parts of the call2service coupling algorithm.

A) *Cost model for service differentiation* is based on service materialization call cost per tuple. The cost is derived from materialization service properties. The cost formula derives from the service interface materialization property metrics, it is expressed per materialization call c as

$cost(c) = W_{QPS} \times QueryPageSize + W_{WS} \times WithDuplicates + W_{CPD} \times CallsPerDay + W_{RPQ} \times ResponsePerQuery$, where W_{QPS},W_{WS}, W_{CPD} and W_{RPQ} are weights associated with each metric, $0f \leq W \leq 1f$.

Table 7.1 exemplifies differentiation between *MovieByTitle* access pattern mapped services {IMDB1, GM1, GM2} considering their service materialization property metrics obtained after each service executed a materialization call, the cost formula assumes equal significant of each property in the calculation $W = 1$. The example also shows newly established order of services on the basis of their cost per tuple.

SI	WithDuplicates	QueryPageSize	ResponsePerQuery	cost(c_i)	OrderedByCost	SI
IMDB{c_i)	1-9/10	1-10/10	1/4500	0.1002		IMDB(c_i)
GM1{c_i)	1 - 5/10	1 - 5/10	1/9000	1.0001		GM2(c_i)
GM2{c_i)	1 - 1/20	1 - 20/20	1/6400	0.9501		GM1(c_i)

Table 7.1: Services materialization metric differentiation.

B) *Cost model for query differentiation* is based on domain value statistics scores. The cost formula derived from domain value statistics for domain value d_n is as follows:

$score(d_n) = W_F \times \sum_{i=1}^{n} freq(d_n) + W_{HR} \times \sum_{i=1}^{n} hRate(d_n)$; where W_F and W_{HR} are weights associated with each score. During the query generation phase queries are ordered by the scores of their values.

Table 7.1 exemplifies computation of $Year$ scores for the $MovieByTitle$ materialization and consequent ordering of expressed queries. The score formula assumes equal significance of each statistics in the calculation, W=1.

YEAR	frequency	harvestRate	score(d_n)	OrderedByScore	Q
1999	1-1/78	1-1/7	1.8442		$q^p < 2003 >$
2003	1-1/56	1-1/12	1.8987		$q^p < 2008 >$
2008	1-1/123	1-1/10	1.8918		$q^p < 1999 >$

Table 7.2: Output domain value metrics differentiation.

7.3.2.1 Optimized Materialization Algorithm

In this section we provide a generic optimization algorithm 8 based on the materialization process defined in Section 4. The algorithm implements the optimization approach by applying the cost model delivered augmentation to *service interface selection strategy* and *query queue selection strategy* parts of the call2service coupling. The algorithm is shown in SPMS materialization scenario with SSQ execution model.

The algorithm assumes an initial, preconditioning phase where (i) for each input domain attribute of the participating service, an initial dictionary *initDict* is provided via the Dictionary input value strategy, (ii) a starting query sequence is initiated by creating queries from the initial dictionaries, (iii) a sequence of service interfaces is instantiated, the sequence features natural order of services as initially their execution cost is unknown.

By loading the initial set of queries, the initial phase of the SPMS materialization ensures the feasibility of the materialization. The algorithm assumes presence of domain-matching attributes in input and output domains, thus, enabling sourcing of the input domain values via the Reseeding input value strategy.

Algorithm 8 Optimized Materialization Algorithm

1: var $\{SI\}$ // sequence of si mapped to feasible_ap
2: var *best_si* // si with best materialization properties., i.e., si with least execution cost incurred
3: var *best_mcall* //mcall containing query expressed by value(s) with highest domain stats scores
4: var $\{C\}$ // materialization call sequence of legal_ap
5: var R_m //wanted materialization
6: var V_I //input domain values set
7: var $feasible\_ap \leftarrow Preconditioning()$ // ap with available queries in the query queue
8: $\{C\} \leftarrow legal_ap.getMaterializationSequence()$
9: $\{SI\} \leftarrow legal_ap.getMappedServices()$
10: //Generic Optimization procedure
11: **while** $\neg Cov_m || \{C\} \neq \emptyset$ **do**
12: $best_{si} \leftarrow \{SI\}.getHeadElement()$
13: $best\_mcall \leftarrow \{C\}.getHeadElement()$
14: $best\_mcall.executeQuery(best\_si)$ //call2service coupling
15: $R_m \leftarrow best\_mcall.Result$
16: // service interface selection strategy augmentation
17: $best\_si \leftarrow cost(best_m call) = best\_mcall.getWithDuplicates() + best\_mcall.getQueryResponseTime() + best\_mcall.getQueryPageSize()$
18: $\{SI\}.put(best\_si)$
19: $\{SI\}.reorderByCost()$
20: // query queue de-queuing strategy augmentation
21: **for** each v in *best_mcall.Result* **do**
22: $score(v) = freq(v) + hRate(v)$
23: $V_I \leftarrow Reseeding(v)$
24: $\{C\}.createNewQueries(VI)$
25: $\{C\}.reorderByQueryQueue()$
26: **end for**
27: **end while**

Algorithm 9 Greedy Materialization Algorithm

1: var $\{SI\}$ // sequence of si mapped to feasible_ap
2: var *best_si*, *win_si* // best_si for sampling phase, win_si winner of the sampling phase used in progressive phase
3: var *best_mcall* //mcall containing query expressed by value(s) with highest domain stats scores
4: var $\{C\}$ // materialization call sequence of legal_ap
5: var R_m //wanted materialization
6: var V_I //input domain values set
7: var $feasible\_ap \leftarrow Preconditioning()$ // ap with available queries in the query queue
8: var $\{C\} \leftarrow legal_ap.getMaterializationSequence()$ // materialization call sequence of feasible_ap
9: var $\{SI\} \leftarrow legal_ap.getMappedServices()$ // sequence of si mapped to feasible_ap
10: var $\{C_{sample}\} \leftarrow \{C\}.getSubset(n)$ // allocates subset of C
11: //Sampling phase
12: **for** each $cin\{C_{sample}\}$ **do**
13: GenericOptimizationProcedure()
14: $\{SI\}.reorderByCost()$
15: $win\_si \leftarrow \{SI\}.getHeadElement()$
16: **end for**
17: // progressive materialization phase
18: **while** $\neg Cov_T || \{C\} \neq \emptyset$ **do**
19: $best\_mcall \leftarrow \{C\}.getHeadElement()$
20: $best\_mcall.executeQuery(win\_si)$ //call2service coupling
21: $R_m \leftarrow best_mcall.Result$
22: **for** each v in $best\_mcall.Result$ **do**
23: $score(v) = freq(v) + hRate(v)$
24: $V_I \leftarrow Reseeding(v)$
25: **end for**
26: $\{C\}.createNewQueries(V_I)$
27: $\{C\}.reorderByQueryQueue()$
28: **end while**
29: // scraping materialization phase
30: **while** $\neg Cov_T || \{C\} \neq \emptyset$ **do** GenericOptimizationProcedure()
31: **end while**

7.3.2.2 Greedy Materialization Algorithm

The optimized call2service coupling algorithm proposed in the previous section, although able to deliver an improvement to materialization efficiency on the whole, may experience performance degradation as the number of services increases, as shown in our experimental results in Section 7.4. To improve materialization scalability for access patterns with high number of services we propose a greedy variation 9 of the original algorithm.

The greedy algorithm is divided into following phases:
(i) a preconditioning phase as in the generic optimization algorithm,
(ii) a sampling phase where a subset of calls is executed and scored against a sequence of service interfaces selected from {SI} in order to establish the best performing service interface win_{si},
(iii) a progressive materialization phase where de-queued materialization calls are coupled exclusively with win_{si} until a predefined threshold coverage Cov_T is achieved, this assumes $Cov_T \leq Cov_m$, (iv) lastly if Cov_m is not reached, a scraping materialization phase is initiated in which materialization calls are again coupled against all services interfaces in {SI} as in generic optimization algorithm. The scraping phase is necessary due to $V_{I,AP} = (V_{I,si,1} \bigcup ... V_{I,si,n})$ where $n = |SI|$, and $V_{O,AP} = (V_{O,si,1} \bigcup ... V_{O,si,n})$, where $n = |SI|$, access pattern properties. A scraping materialization phase is envisaged as a last resort attempt to reach Cov_m, despite the potential efficiency decrement.

7.4 Empirical Study

We have designed our empirical study to be relevant to the underlying SeCo search requirements. As described in Chapter 4 access patterns and their services are organized in SDF and linked via their semantic relations to Domain diagram (DD). This is further put into service of SeCo exploratory search paradigm [Bozzon 2011b] where users can express their queries directly upon the concepts known (registered) to the system such as jobs, restaurants, theaters, shops etc.

The exploratory query interaction paradigm proceeds as follows. The user

starts by selecting one of the available objects, and submits an "object query" to extract a subset of object instances. For example, a user could choose a "jobs" object, and ask for "java" jobs in some "salary range" in the "CBD" area of "Auckland City", or choose a "shop" object, and ask for "toys" merchandises in the "West area" of "the Greater Auckland" in a given "price range". At any stage, users can "move forwards" in the exploration, by adding a new object to the query, starting from the connections available in the DD and from the objects that have been previously extracted.

By having this type of SeCo Search in mind we have based our experiments on a few common concepts, their access patterns and mapped services registered in the SDF.

We evaluate the performance of the proposed optimization model and the algorithms outlined in Section 7.3.2 in terms of coverage over the number of materialization calls efficiency E_C, and coverage over period of time efficiency E_T. The goal of the experiments was to establish the materialization efficiency increase achieved by performing optimized materialization in comparison to a non-optimized solution outcome.

We considered a single pattern multi service scenario (spms) in different materialization call sequence execution models as described in Section 4.5 of Chapter 4. For each materialization algorithm experiment we compared optimized serial execution models performances with parallel execution model, and random non-optimized execution model performance.

The empirical study was performed by considering the access pattern *shoppingItemByLocation* which takes shopping item location and item type as input attributes, where $I_{shoppingItemByLocation} = \{ItemType, Location\}$ and $O_{shoppingItemByLocation} = \{ItemName, Price, SaleType, Location\}$. The item type input dictionary was populated by a dictionary input strategy which used a static, pre-populated list of input values $Dict_{ItemType} = \{'books', 'toys', 'clothes'\}$. The *location* input attribute was populated by using a reseeding strategy via the domain-matching attribute *Location* in the output.

To perform the evaluation, we created a master database composed of $\tilde{1}$00K listed items from several existing on-line US auction and shopping sites.

A domain of US postcodes found in the collected shopping items database was used as location input attribute domain. The domain size of the postcodes reseeding output attribute was approximately 8000, a randomly selected subset of which was used as an initial seed dictionary. For each of the materialization (result generation) strategies 30 runs were performed; to avoid biases in the evaluation, the location input attribute conforming to the reseeding input strategy was initialized at each run by a randomly selected subset of Location dV as $initDict_{Location}$ of size 50.

Provided are several initial first chunk (page) queries: q_1^1<'books','43314'>, q_2^1< 'books', '23857'> and q_3^1<'books', '80002'>, q_4^1<'toys','43314'>, q_5^1< 'toys', '23857'> and q_6^1< 'toys','80002'>, q_7^1< 'clothes', '43314'>, q_8^1< 'clothes', '23857'> and q_9^1< 'clothes', '80002'>.

The resulting materialization coverage increase had been averaged between runs for each strategy and observed in intervals of 10% with regards to the overall materialization size.

Several experimental source databases were created as a subset of the master database. Each source consisted of a varying ratio of shared and unique items, thus, maintaining different levels of heterogeneity between sources. Each source was allocated a time value that was used as an average for deriving random, normally distributed query response times - Table 7.3 for each materialization call variance $\sigma = 0.2$.

SI	Unique Tuples (%)	Response Time mean(ms)	Max Page Size	Max Result Size
ShopItemByLoc_1	80	5100	10	100
ShopItemByLoc_2	60	4000	20	100
ShopItemByLoc_3	40	8200	10	100
ShopItemByLoc_4	40	6500	30	100

Table 7.3: Relevant properties configuration of the experimental source databases.

Experiment Setting We ran our experiments on a Dell Precisions T1500 workstation under Ubuntu Linux 12.04 LTS, Intel Core i7-870 CPU, 8GB of DDR3 SDRAM, and NVIDIA Quadro FX 580 graphics card and 320GB HDD. For the purpose of the experiment the source databases were fronted by a REST web service that interfaced the search process. The materializer module ran under Apache Tomcat 7.3 web container, the materialization was persisted into PostgreSQL 8.3 DBMS. The materialization module, the SeCo search engine and the database were hosted on its own Oracle VirtualBox based Ubuntu virtual machine. The experimental source databases were hosted on PostgreSQL 8.3 DBMS under their own Ubuntu virtual machine with the search REST web service deployed to Apache Tomcat web container.

7.4.1 Coverage Efficiency for serial single queue (SSQ) execution model

Firstly we observe performance of the optimization algorithms in the serial single queue (SSQ) execution model. Figure 7.10 delivers two sets of results 'Service optimization' where just service interface selection strategy optimization were applied and 'Full Optimization' where both service selection and query queue prioritization were applied.

As expected the single queue nature of the execution process obscures the effect of the domain value characteristic metrics as analysis is performed against results obtained across all participating services contained in the single query queue. Figure 7.10 demonstrates this behaviour, as inclusion of the query queue optimization - Full Optimization fails to deliver considerable improvement. The greedy algorithm performs better due to the prioritization of the single 'best performing service' throughout the process, thus allowing the domain characteristics analysis and consequent query queue ordering to be performed against the single source. The upper segment of the Full Opt result shows mild improvement in the obtained coverage due to inclusion of the query queue optimization. This also shows that the prioritization of the 'popular' nodes in the domain distribution delivers small but constant improvement to the efficiency of the materialization process.

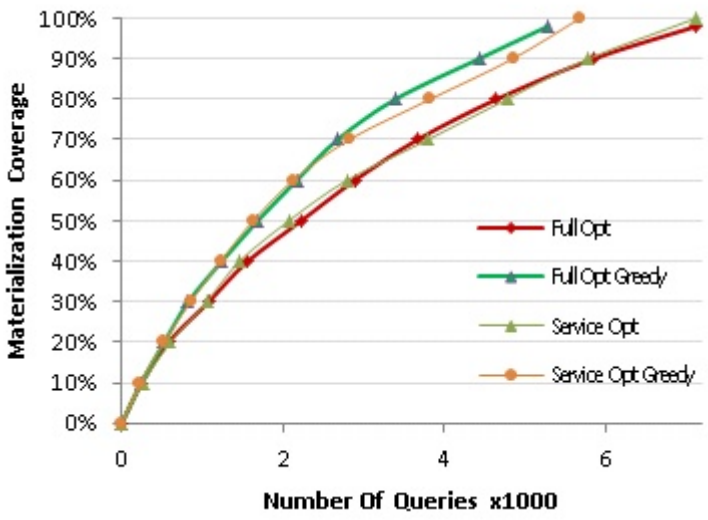


Figure 7.10: SSQ Service Optimization vs Full (service selection and query queue) Optimization.

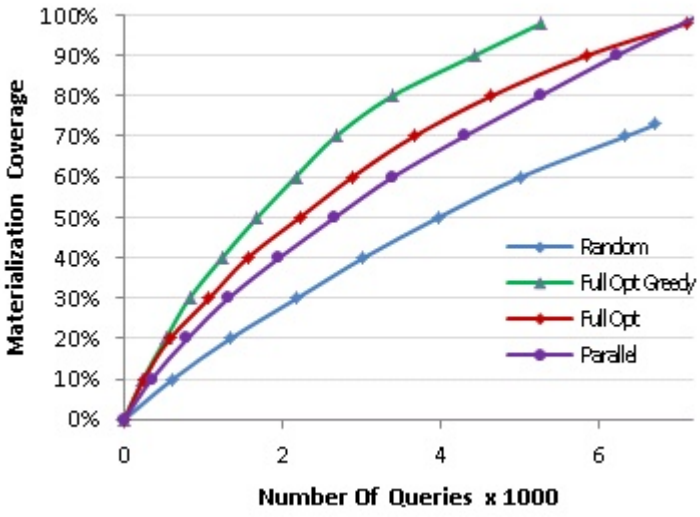


Figure 7.11: SSQ performance Full (service selection and query queue) optimization results.

Figure 7.11 provides the result of the SPMS materialization scenario obtained by performing non optimized materialization process 'Random', two optimized processes Full Opt and Full Opt Greedy and a parallel non optimized process. The benefits of the optimization approaches were evident as both optimization algorithms consistently achieved the best coverage with the least communication effort. The generic optimization algorithm (Opt) expressed 1.8-fold coverage E_C efficiency increase at 50% of the materialization, while the greedy optimization algorithm (Greedy Opt) shows a 2.4-fold efficiency increase. Parallel process shows 1.4 times efficiency increase in comparison to the random process. The efficiency at 70% coverage remains increased at similar levels for both generic Opt and Opt Greedy algorithms.

This efficiency increase illustrates the importance of the sampling phase of the Greedy algorithm as this phase clearly elects the best service in terms of its performance qualities - low response times, low duplicates, high result size, and reseeding domain characteristics. This further enables the progressive phase to efficiently reach the wanted materialization coverage by using just the elected, sampling phase winner service.

The parallel process fair result exemplifies the nature of the parallel execution sequence where all services are materialized in their own data acquisition phases, thus, effectively performing several individual materializations in parallel.

Worth noticing is the failure of non-optimized process to achieve 100% coverage. This illustrates the benefits of the optimization approaches versus issues outlined in section 7.1 - low reseeding domain coverage, thereby providing inadequate input dictionary, high duplicate levels and small result sizes. It also stresses the drawbacks of the serial single queue execution model where a once executed a query is unreachable with respect to other participating sources.

We have clearly demonstrated that domain distribution based query prioritization together with service optimization deliver significant improvement to the efficiency of the materialization process and delivers full accomplishment of the materialization task.

7.4.2 Coverage Efficiency for serial multi queue (SMQ) execution model

As a contrast to the SSQ model, serial multi queue execution (SMQ) allows for query queue prioritization to be executed for every participating service. This in turn allows for domain distribution of each service to be fully exploited in the optimization process. This was evident in the optimization performance comparison between service profile based optimization versus the optimization utilizing both service selection strategy and query queue de-queuing strategy optimization methods, as shown in Figure 7.12.

The influence of the domain analysis was particularly pronounced in the 50% to 80% of the materialization coverage i.e., once enough of the domain was discovered to make an impact on consequent query queue prioritization. The positive effect diminished after coverage exceeded 80% when all approaches deteriorated. This shows that after the coverage reaches a certain level, most returned tuples overlap with the previous results. In the literature this is referred to as the 'low marginal benefit' phenomenon in query-based database crawling [Wu 2006].

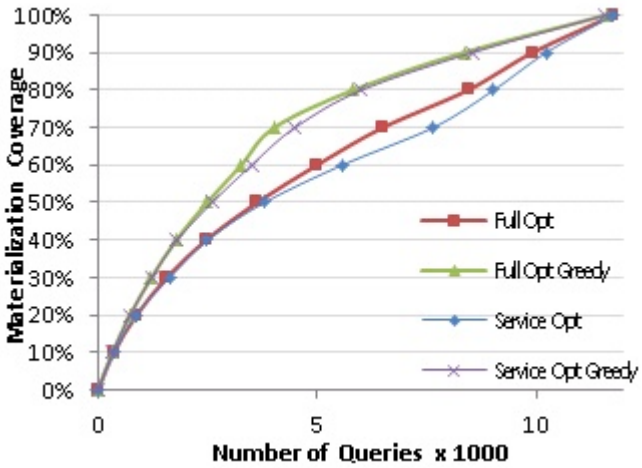


Figure 7.12: SMQ Service Optimization vs Full (service selection and query queue) Optimization.

Figure 7.13 shows coverage efficacy increase for generic optimization Full

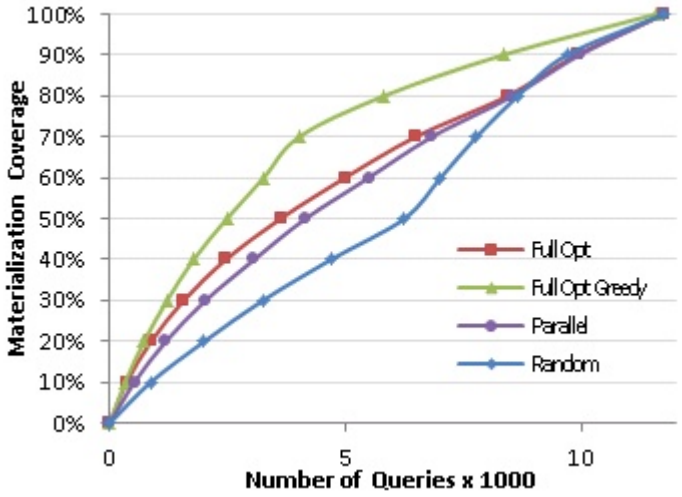


Figure 7.13: SMQ Full (service selection and query queue) optimization performance results.

Opt and greedy optimization Full Opt Greedy algorithm. At 50% coverage the efficiency increase was 1.7-fold for generic optimization and 2.5-fold for greedy optimization algorithms when compared to the random, non-optimized process. However the generic optimization process decreased in efficiency and meets the non-optimized solution at 75% of the achieved materialization. This matched our expectation that due to the exhaustion of the expressive 'popular' input domain values in individual source queues as well as the general drain of unique results within sources the efficiency would decrease after a certain percentage of the coverage was exceeded.

Parallel execution model achieved steady performance of 1.5 times performance increase, just below the generic optimized algorithm performance. It showed the same efficiency decrease at 75% of the coverage as the Opt algorithm. The greedy optimization still maintains a two-fold efficiency increase at 80% of coverage. It gradually decreased once it entered the scraping phase i.e., reintroduced all sources to the process.

The experiments also outlined the difference in *efficiency scale* of the execution models. The SMQ execution maintained the same queries vs. coverage ratio up until 50% of the coverage. The scale of the MQ execution drastically increased after 70% of the coverage was exceeded as queries were executed

against all participating services in the scrape phase as opposed to SQ execution where queries once executed were unavailable to the other sources.

In terms of efficiency scalability our results suggest better suitability of the proposed approaches in the context of smaller, output domain condensed data sources where materialization task is possible against limited number of services. The introduction of sparsely populated, duplicate ridden sources leads to the degradation of the process efficiency due to the increase of materialization calls issued to reach the wanted coverage.

7.4.3 Time efficiency for SSQ and SMQ execution models

Figure 7.14 illustrates the increase in E_T efficiency of the materialization process achieved by applying generic and greedy optimization algorithm to serial single queue model (SSQ). The time efficiency increase caused by generic optimization algorithm Opt was 3-fold and almost 4-fold for the greedy variant of the optimization algorithm.

The parallel process, as expected matched the 4-fold efficacy increase of the greedy algorithm, even surpassing it in the final 20% of the materialization.

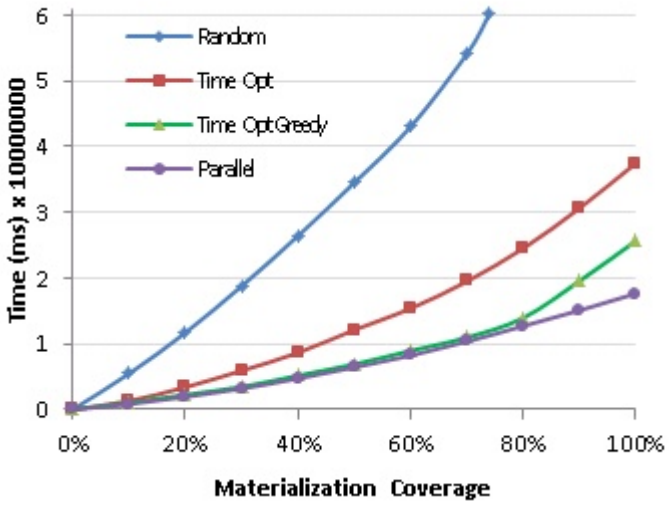


Figure 7.14: SSQ Time optimization performance result.

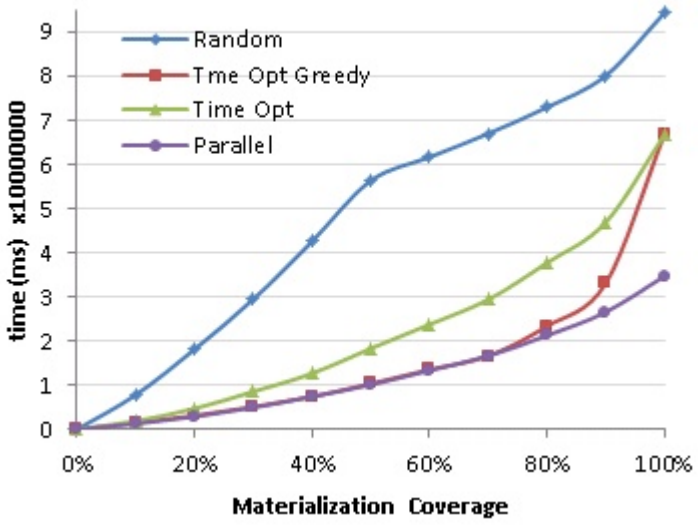


Figure 7.15: SMQ Time optimization performance result.

Time dimension efficiency increase of SPMS materialization process with SMQ execution model is shown in Figure 7.15. It confirmed and amplified the pattern of efficiency increase observed in the single queue execution mode experiment. The generic optimization increased E_T efficiency 3-fold at 50% coverage while the greedy variant increase is 5-fold. This trend was maintained at 80% coverage, gradually decreasing towards the full materialization due to the general unique tuples exhaustion across the involved sources.

As expected, parallel execution model matched the Greedy Opt algorithm and surpassed it in the last 20% of the materialization.

7.4.4 Live large data source evaluation

To further assess validity of our optimization approach we executed a materialization of the stack exchange group of sites [SOCareers 2014, CareerJet 2014] with a focus on job related search. The aim of the experiments was as in the previous case to establish the materialization efficiency increase achieved by performing optimized materialization in comparison to a non-optimized solution outcome.

We considered a single pattern multi service scenario (spms) in different

materialization call sequence execution models as in the previous experiment. For each optimization algorithm experiment we compared optimized serial single queue (SSQ) and serial multi queue (SMQ) execution model performances with parallel execution model, and random non-optimized execution model performances.

The empirical study was performed by considering the access pattern $jobByKeyword\&Location$ which takes a job location and a job search keyword as input attributes $I = \{jobKeyword, Location\}$ and retrieves $O = \{jobTitle, jobType, JobProvider, Location\}$.

The *job search* keyword input dictionary was populated by a dictionary input strategy that used a static, pre-populated list of input values $Dict = \{'Javaprogrammer', 'Sysadmin', ..., 'Civilengineer'\}$ of size 50. The *location* input attribute was populated by reseeding from the *location* attribute in the output domain.

We provide few initial first chunk (page) queries q_1^1<'Java programmer', 'New York'>, q_2^1<'Web designer', 'Chicago'> and q_3^1<'Dentist', 'Houston'>. As both services returned paginated results we ran paged queries until the result set execution, that is an empty result page was reached.

We performed 20 materialization runs over a period of time. The materializations were performed until the process exhausted all the generated queries. The size of the queried data sources was unknown in advance due to the constraints imposed by the service level agreement of the materialized web sources, thereby it was impossible to pre-set the wanted materialization size. The final materializations size on average was close to one million tuples.

Experiment Setting We ran our experiments on a custom built workstation under Ubuntu Linux 13.04 LTS, AMD FX(tm)-8320 Eight-Core Processor, 3500 Mhz, 4 Core(s), 8 Logical Processor(s), and NVIDIA GeForce GTX 650 graphics card and 1TB HDD. The materializer module ran under Apache Tomcat 7.5 web container, the materialization was persisted into PostgreSQL 8.3 DBMS.

The results of serial single queue (SSQ) performance efficiency E_C matched our expectations. The full greedy optimization outperformed other approaches from 1.5-fold in case of generic optimization to 3-fold in the case of random

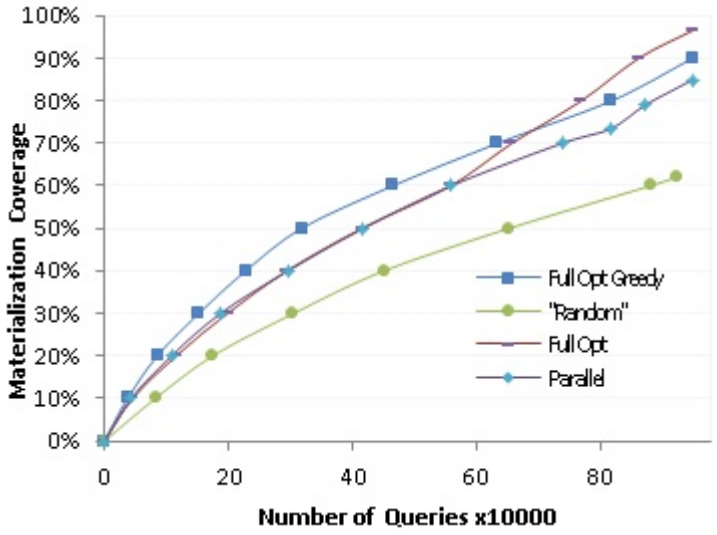


Figure 7.16: SSQ Full optimization performance results.

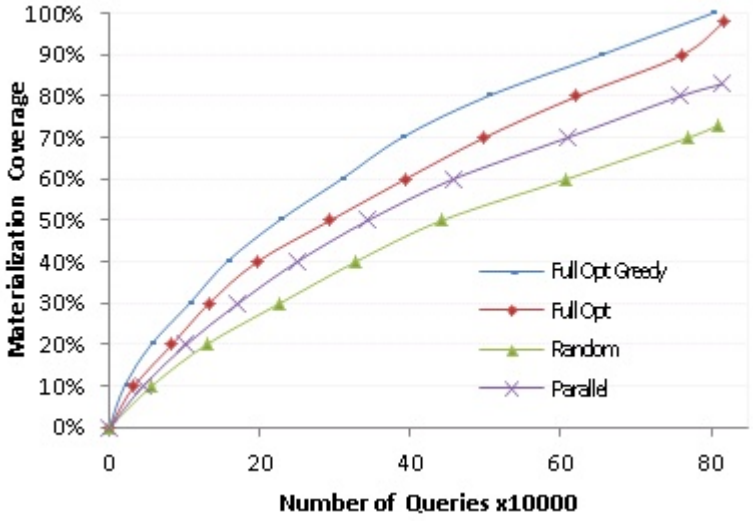


Figure 7.17: SMQ Full optimization performance results.

non-optimized experiment result. Interesting behaviour appears in the upper segment of the Full Opt result where a sudden surge in coverage efficiency may be observed. We explain this feature by the nature of the single queue execution where one of the services 'wins' and holds its lead until the queue is exhausted while the greedy algorithm in this segment is already in the scraping phase finalizing 'leftover' services of lower coverage reach.

The coverage efficiency of the serial multi queue (SMQ) proved the greedy algorithm to be a reliable performer, winning over other approaches. In this experiment the multi queue nature of the execution model maintained stable performance over all segments of the materialization execution.

Figures 7.18 and 7.19 provide further evidence of the increase in E_T time efficiency of the materialization process achieved by applying generic and greedy optimization algorithm. Evidently, the unpredictability of the Internet environment plays a role as the E_T time efficiency increase is not so pronounced in the serial single queue model execution as it is in a case of more controlled environment in the first set of experiments. The parallel execution widened the performance gap by more than 20% in all segments of the materialization coverage. This also demonstrates a limitation of the single queue execution model - once executed against a not so productive service a potentially expressive query is unrecoverable for the other services.

Nevertheless, in the case of serial multi queue model the Time Opt Greedy algorithm closed the performance gap with the parallel execution to around 10% in most of the materialization coverage.

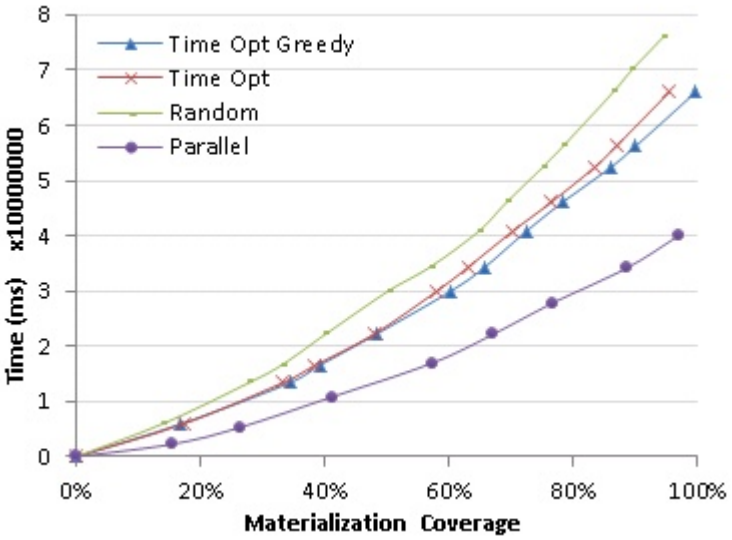


Figure 7.18: SSQ Time optimization performance result.

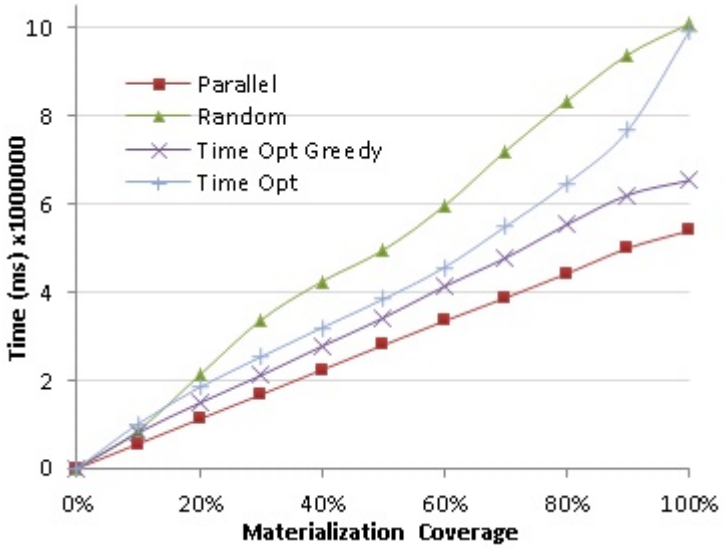


Figure 7.19: SMQ Time optimization performance result.

7.5 Chapter Summary

In this chapter we identified and formally defined factors acting detrimentally to the web source data materialization. We identified and formally defined a set of web data source properties relevant to the materialization process performance. We also established a set of domain value characteristics that significantly influence materialization query expressiveness. We proposed a set of metrics and a subsequent cost model to be used in the optimization algorithms tackling aforementioned detrimental factors.

We demonstrated the effectiveness of our optimization approach over a series of experiments. The experiments were conducted considering several materialization execution models against a medium size data set realistically mimicking 'real world' situation. The results proved the effectiveness of the approach as the improvement of materialization efficiency in most cases were close to 3-fold when compared to the non-optimized materialization algorithms results.

We further demonstrated the validity of the optimization approach by performing a series of materializations of the actual web data sources that resulted in obtained materialization size close to one million tuples. The performance increase of E_C efficiency was analogous to the first set of results. The E_C efficiency increase was present in the live experiment results although not so pronounced as in the more controlled environment of the first set of experiments.

Most of the approaches tested were able to materialize the first 50% of the given materialization task in an efficient and timely manner. However, the efficiency of the algorithms performed in the second parts of the materialization process varied. The greedy algorithm performed optimally in both single queue and multi queue execution, however with decreased efficiency as the scale of the process increased in multi queue scenarios. In all tests domain structure driven query queue optimization achieved the best results once 50% or more of the materialization coverage were exceeded, that is, optimal domain distribution knowledge was obtained. Given this empirical evidence it is fair to conclude that if the domain distribution of reseeding values were known

a priori the query queue based optimization would deliver a more significant optimization result.

Our optimization goal was to optimize the materialization process while maintaining the right leverage between performance and coverage of the materialization task.

We intentionally left the 'right leverage' indicator loose as it is hard to predict the right in the search task context. Sometime we materialize just a fraction of the source volume and that is sufficient to feed the search process until a more responsive and result generous service is found. Sometime even with most of the source materialized the incoming query might request results from the remaining - not materialized - source subset, thus rendering the materialization effort futile. However, in the context of the given materialization task the empirical study has proven that a considerable materialization optimization level has been achieved. In both SQ and MQ materialization experiment full optimization have achieved 3-fold and 2-fold improvement in 50% and 75% coverage of the materialization task over un-optimized, random materialization run. We expect this to be a valuable contribution and the right leverage for most of the search scenarios where materialization tasks for the specific, focused domain are required.

The open question remains how the proposed approaches perform against very large sources with 10 - 100 million tuples sizes. The main obstacle is the availability of such sources as they are typically fronted by fairly conservative SLAs. This makes our testing routine pointless as the time necessary to achieve one materialization test might stretch over several months.

Chapter 8

Limitations and Future Work

8.1 Introduction

In this chapter we address the limitations of the presented research. We discuss their impact to the materialization solution and outline a possible way forward in overcoming these limitations.

We discuss shortcomings in the areas of materialization optimization, query containment, and materialization maintenance. Moreover, we present a possible way forward in terms of possible solution and research continuation.

8.2 Research Limitations and Future Lines of Work

Whilst we have provided major contributions to the materialization formulation problem by characterizing web materialization dimensions, providing a feasibility analysis model and delivering materialization optimization, much room still remains for further improvement of materialization building blocks as defined in Chapter 4.

Query containment as one of the main concepts behind view materialization has not been given enough focus in this work. The idea of relaxing materialization query to increase the yield of the obtained results set is a paramount for the improved optimization of the materialization process. At present, materialization process is still constrained to equivalent rewrites of the provided queries. An addition of the query expansion algorithm to the optimization process would deliver ultimate optimization effect, as this would enable materialization of potentially whole domains with minimum input dictionary. For instance, query expansion from 'Select Thai restaurants in Herne

Bay, Auckland' to 'Select all restaurants in Auckland' is set to achieve full Auckland restaurant domain in just one issued query. In the remainder of this section we will present a potential way of delivering this improvement of our research.

One further limitation of this research is the lack of a dedicated sampling routine. A materialization attempt may prove to be futile if a selected service is unresponsive, duplicate-saturated or simply not conforming to the provided initial dictionary. Even though presented optimization algorithm aims to omit such services during materialization execution this may still prove to be inadequate. Concise materializations are required to satisfy needs of domain focused, specific results driven search scenario such as exploratory search. To limit such risk and provide desired level of precision the materialization process needs to be enhanced by addition of a remote data source sampling routine. Naive implementation of sampling routine is based on the SDF Domain Diagram contained entities and their focuses. For instance, for the given set of access patterns, their focus entries and defining attributes are considered. In case of access pattern *RestaurantByCuisine* we consider an entity with focus *Restaurant* and attribute *Cuisine*, or in case of access pattern *TheaterByLocation*, entity with focus *Theatre* and attribute *Location*, to perform quick sampling exploration of the available services. Thus defined 'sampling heuristics' is further employed in some of the domain graph sampling routines such as *Random graph walk* or *Random Jump* sampling [Leskovec 2006] to obtain reduced yet representative sample of the domain graph for the given web sources.

The topic of materialization maintenance has also been left out of the scope of the thesis due to its own complexity and size.

In the following subsection we provide some thoughts on a possible way forward in regards to materialization optimization and query expansion as well as also our view of a potential materialization maintenance procedure.

8.2.1 Enhanced Optimization by feasibility analysis variables

One of the future lines of work lies in expanding the materialization optimization routine by considering optimization enhancements delivered through the feasibility analysis model.

As feasibility solution analysis relies on the methodology based on graph properties of the implemented model, the measures described in this section are based on notions defined in the graph theory [Easley 2010].

8.2.1.1 Shortest Distance

Shortest distance is based on the notion of shortest path from the initial marking M_0 in a given reachability set R_{mp} at one end and the executable set of transitions L_{mp} as imposed by the feasibility solution at the other end. The idea is to prioritize access patterns that satisfy more dependencies. We do the prioritization by taking the count of transitions in L_{mp} executed to reach a certain place in R_{mp}. As illustrated in Figure 8.1 initial marking $M_{0mp} = 1,0,0,0,0,0,0,0,0,0,0$ reaches sequence $M_{imp} = 1,1,1,0,1,1,1,0,1,1$ following a sequence of transition firings - Tc_{ap1},

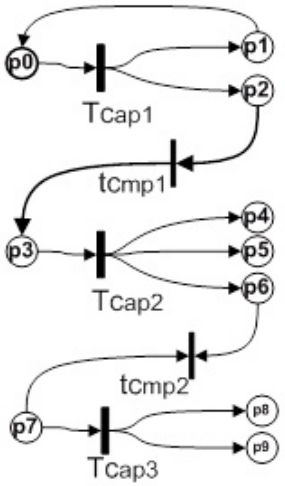


Figure 8.1: Shortest Distance metrics example.

tc_{mp1}, Tc_{ap2}, tc_{mp2}, Tc_{ap3}. $R_{mp}(M_{0,mp}) = p0, p1, p2, p4, p5, p6, p8, p9$, while $L_{mp}(M_{0,ap}) = Tc_{ap1}, tc_{mp1}, Tc_{ap2}, tc_{mp2}, Tc_{ap3}$.

To reach p8 and p9 it is required to fire five transitions - $Tc_{ap1}, tc_{mp1}, Tc_{ap2}, tc_{mp2}, Tc_{ap3}$, while to reach p4, p5, p6 - three transitions are fired - $Tc_{ap1}, tc_{mp1}, Tc_{ap2}$, and just one transition - Tc_{ap1} to reach p1 and p2.

Following the above demonstration we define *ShortestDistance* metric, the prioritization measure based on services position in the materialization feasibility solution.

Definition 8.1. The materialization cost applied by the *ShortestDistance* property to the materialization R_M is defined as: $cost[q_p^k, r_p^k] = NoOfTransitions/C$, where *NoOfTransitions* is the size of L_{rm} where $1 <= NoOfTransitions$, and C is a constant, arbitrary value, determining the range of the cost metrics.

For instance, if *NoOfTransitions* leading to the service in the feasibility solution is 5 and C=10 then the service materialization call cost is 0.5, if NoOfTransitions=1, the cost is 0.1.

8.2.2 Query expansion

Current research has not addressed the query expansion concept to the desired degree as suggested in the introduction to this thesis. Hence, another limitation of the current research and potential area for improving web materialization efficiency in terms of query containment is semantic query expansion [Bhogal 2007]. One possible way of query expansion is illustrated further below.

In SeCo service description framework context an issue of query expansion and web data materialization as a main area of interest can be approached as follows.

In the domain diagram definition there is a space for enhancement in the definition of attribute types. The enhanced data type will enable us to define the semantic relations between data domains, their attributes and repeating

groups. Semantically aware attributes will provide us with the logical connections between different domains not just at the conceptual level but also at the logical and physical levels thus enabling us to reformulate, expand or shrink multi domain queries on the fly at the run-time.

In order to do so we will define a complex type that describes Domain attributes as:

Data Type the actual type of the attribute; it consists of: *type* field - String, Number, or Boolean; *isRanged* field - Boolean, true if ranged, false otherwise; *Range* field - the range of the attribute if applicable;

Semantic Type a type describing attribute semantics, it consists of: *supertype field* - attribute's predecessor in the logical sequence (ontology); *Subtype* field - attribute's successor in the logical sequence (ontology); the *ontology identifier* field;

Rank describes the ranking abilities of the attribute, it consists of: *isRankable* field - Boolean true if rankable, false otherwise; *Ranking function* pointer(s) - pointer(s) to ranking function(s) applicable to the attribute.

AttributeType

Semantic Type(Predecessor, Successor, OntoID)

DataType(Type(String, Number, Boolean), isRanged, Range)

Rank(isRankable, RankFunctions[])

If we consider attributes ZipCode and Postcode found in two access patterns in different domains, assuming their semantic connectivity and postcode being a predecessor of the zip code, their complex type is defined as:

Postcode

$SemanticType : (null, ZipCode, CodeOnto)$

$DataType : (Number, true, 0..n)$

$Rank : (true, [f_1, f_2, ..., f_n])$

ZipCode

$SemanticType : (Postcode, null, CodeOnto)$

$DataType : (Number, true, 0..n)$

$Rank : (true, [f_1, f_2, ..., f_n])$

Once we have the complex type that reflects semantic relations between attributes defined, we apply it as follows to the SDF levels.

Conceptual Level If we apply the semantically enhanced attribute at the domain diagram - conceptual level we observe the following:

Provided we have two Domains

Restaurant(Name, Location(Street, Suburb, City, ZipCode, Region, Country), Phone, Url, Rating, Category(Name))

which describes a class of web objects containing all information about restaurants.

GeographicLocations(Longitude, Latitude, Number, Street, Suburb, City, Postcode, Region, Country, Continent)

which describes a location geographically.

Further, we define logical sequence or ontology and present it with the semantically aware - complex type attributes as proposed above.

GeoLocoOnto($Street \rightarrow Suburb \rightarrow City \rightarrow Province \rightarrow Country$)

Then, we apply it to the appropriate attributes in the domain diagram; in the Restaurants domain **Street, Suburb, City, ZipCode, Region, Country** and **Number, Street, Suburb, City, Postcode, Region, Country** in the GeographicLocation domain.

Therefore, after the complex type application to the attributes of both domains with the above described complex types, we end up with a semantic cross section between these two domains. If we describe it via a hyper-graph we see that each attribute is a node and each hyper-edge a semantically connected view between these domains.

The definitions of the boundaries (or frontiers) of the multi-domain cross-sections are contained within the Rank of the complex type where ranking capabilities of the attributes and corresponding rank functions are described.

In the following example we define two Domains with the TimeLine ontology applied to their attributes' complex types:

Movies(Title, Director, Score, Year, Language, Genres(Genre), Open-

ings(Country, Date), Actors(Name))
and **TemporalLocation**(Second, Minute, Hour, Day, Month, Year, Century)

Then we define and apply the complex type to TimeLine ontology
TimeLineOnto($Year \rightarrow MonthDay \rightarrow Hour \rightarrow Minute$)

Clearly, there is a semantic connection between the release year, the opening date of the Movies domain and the TemporalLocation domain via the TimeLine ontology. Furthermore, the Openings repeating group also contains a semantic connection between its country attribute and the corresponding attribute in the GeographicLocation domain, as described above, via the GeoLoco ontology.

Hence, there are three different domains semantically connected and if represented by a hyper graph, its nodes denote attributes and hyper-edges describe their semantic connections (views), and the frontier of the cross-sections is defined by the ranking functions present in the complex type.

Logical Level We observe a query about a particular restaurant identified by its location attributes 'suburb' and 'zipcode' to which we apply an access pattern that is associated with the above described *Restaurant* and *GeographicLocation* domains.

If there is a query that looks for a particular restaurant with suburb and zip code as input attributes then the query expansion will be performed by sourcing all the restaurants in the suburb's city and using all the available zip codes or simply a wild card. Worth noting is, even though the access patterns are related to particular domains, that there is a semantic connection defined and present between Restaurant and GeographicLocation via GeoLocoOnto ontology and contained in the attributes complex type definition.

DEFINE QUERY RestaurantPlan($X:Suburb, $Y: ZipCode) AS
SELECT R.
FROM Restaurant (iSuburb: $X, iZipCode: $Y) AS M USING YELP_Wrapper

In this case input attributes' complex types are defined as:
Suburb $SemanticType : (City, Street, GeoLocoOnto)$ $DataType$:

$(String, false, null)$ $Rank : (true, [f_1, f_2, .., f_n])$

ZipCode $SemanticType : (Postcode, null, GeoLocoOnto)$ $DataType : (Number, true, 0..n)$ $Rank : (true, [f_1, f_2, .., f_n])$

Further, in order to resolve the actual value of the semantically expanded suburb type we construct a query resolving the super-type actual value as shown below. This assumes that there is a set of domains in the domain diagram and an access pattern registered that is capable of handling this expansion i.e., the GeographicalLocation described above and semantically bound to the Restaurant domain via GeoLocoOnto ontology.

The query below utilizes two access patterns, one for Restaurant domain and another one for GeographicLocation domain.

AP_Restaurants

$I = Location.City, Location.ZipCode, Category.Name$

$O = Name, Location.Street, Location.Suburb, Location.Country, Phone, Url, Rating_R$

AP_Location

$I = Location.City, Location.ZipCode$

$O = Location.County, Location.Country, Location.Street, Location.Suburb$

NB. The subscript R denotes a ranking attribute in the output domain.

First, the query is resolved as follows:

DEFINE QUERY RestaurantPlan($X:Suburb, $Y: ZipCode) AS
SELECT R.
*FROM Restaurant (**iCity: Suburb.Predecessor, iZipCode: ZipCode.getRangeAll**) AS M RANK BY iRating:Rating.isRankable.getFunction(i)*

Then expanded into the query below it shows the expansion iCity with interpolation:

DEFINE QUERY RestaurantPlan($X:Suburb,$Y: ZipCode) AS
SELECT R.
FROM Restaurant (iCity(Select City from Cites Where Cities.suburb=$X:Suburb.getValue, iZipCode: ZipCode.getRangeAll as C), iZipCode:
ZipCode.getRangeAll) AS M RANK BY iRating:Rating.isRankable.getFunction(i)

Obviously, the expansion mechanism can be applied to all layers of the ontology contained within types and can go both ways. Hence we can focus or blur the result set based on the given semantic relations as required. Or, if described using a layered cake analogy by scrolling up and down the contained ontology we add or remove flavours like we add or remove layers in a layered cake.

The expansion is driven by the logic that String attributes are expanded by a predecessor and Number attribute by max range. In other words, Strings expand semantically into their predecessors or condense to their successors i.e., increase or decrease semantic range, numbers expand by its numerical value to max value or wild card.

Physical Level As represented at the service interface level, there is a bottom up relation between two service interfaces, two access patterns and two different domains via a semantic connection contained within the attributes' complex types.

Service Interface *YelpRestaurants* resolves access pattern AP_Restaurant against a Yelp.com based web service.

Restaurant("YelpRestaurants", AP_Restaurant, http://yelp.com)
Service Interface *USLocations* resolves access pattern AP_Location against a prototype geolocator.com web service specified for US based locations.
GeoLocation("USLocations"' AP_Location, http://geolocator.com/us)

The query is expanded into

DEFINE QUERY RestaurantPlan($X:Suburb, $Y: ZipCode) AS
SELECT R.
FROM Restaurant (iCity(Select City from Cites Where Cities.suburb=$X:Suburb.getValue, iZipCode: ZipCode.getRangeAll as C USING USLocations), iZipCode: ZipCode.getRangeAll) AS M USING YelpRestaurants

Finally, we obtain all the restaurants for the given city that relates to suburb and all the values of the Zip for the given Zip type.

In this way we have obtained a view semantic expansion by location of a query and are capable of answering the query:

DEFINE QUERY RestaurantPlan($X:Suburb, $Y: ZipCode) AS SELECT R. FROM Restaurant (iSuburb: $X, iZipCode: $Y) AS M*

8.2.3 Materialization maintenance

As proposed by [Gupta 1995] and transposed to the web data context, view maintenance is a process of updating materialized web data source in response to changes in the underlying web source. As materialization is proven to be performance and time-wise expensive process we propose our future work on an incremental maintenance procedure based on certain heuristics. The heuristics are based on meta-data availability of the materialized web sources and consequently lead to two maintenance processes as a way forward.

8.2.3.1 White Box Metrics Description

Here the web source provider exposes the data describing the data source, data structure, and updates over time. Based on this information we observe coverage, data accuracy and currency [Peralta 2006] as three main data quality metrics on which we derive incremental materialization maintenance heuristics.

Coverage relative to the materialization (C) - as defined in Chapter 4 section 4.3.1, materialization coverage denotes the ratio between the numbers of tuples discovered in materialization R_m to the total number of tuples in the data source.

Currency (Crn) - is the percentage of the data tuples (Odt) in the materialized data set that has older timestamps than their equivalents (Ds) in the external service data set, where **Odt** is a number of obsolete data tuples in the materialized data set and **Ds** is the external service data size.

Accuracy (Acr) - an average result set difference, expressed as a percentage obtained against the actual source and materialized data with randomly generated queries from the discovered input values.

A result set difference is measured as the number of different tuples found when two result sets are compared.

Metric threshold	Selective MA	Full MA	Full MA Fallback
Coverage $< 10\%$		Yes	
Coverage $> 10\%$			Yes
Currency $< 10\%$	Yes		
Currency $> 10\%$		Yes	
Accuracy $< 10\%$			
Accuracy $> 10\%$	Yes		

Table 8.1: User Driven Maintenance Procedure sample heuristics.

User Driven Maintenance Procedure We briefly introduce a user driven maintenance procedure that does not establish a firm criterion for the actual maintenance but instead provides a flavour of potential maintenance triggers and possible actions imposed by these triggers.

For each metric there is an initial **threshold** defined (e.g., 10%); if this threshold is reached an appropriate procedure involving the data acquisition process is invoked as presented in Table 8.1. **Selective MA** - materialization procedure is invoked at query level, replacing only results affected by the triggering metric, that is, changed data or expired data.
Full MA - materialization procedure is invoked, refreshing the total materialization data using original strategy/plan.
Full MA Fallback - materialization procedure is invoked, refreshing the total materialization data using next available feasible solution.

8.2.3.2 Black Box Metrics description

Here the web source provider SLA policy obscures the data source meta-data. Consequently, we base the materialization heuristics on derived metrics such as freshness and access frequency [Cambazoglu 2010].
Freshness - to assess freshness of results, we use the average age of a hit (stored result set access): the difference between the time of the hit and the time of the last update.

Access Frequency - number of hits by result set, over a period of time.

Naive Maintenance Procedure The policy is driven by a self adjusting TTL (time to live) value derived by observing timestamps in the materialized data and the access frequency for each materialized tuple. We chose to give higher priority to hotter and older queries. That is, queries that appear more frequently (hot queries) and that have been in the storage longer (older queries) have a higher priority for refresh.

8.3 Chapter Summary

In this chapter we acknowledged and discussed some of the most pressing limitations of this work. Particular focus has been given to several ways forward in terms of further materialization process optimization, query expansion and a possible way of implementing materialization maintenance.

In the next chapter we will conclude this thesis, by briefly presenting a summary of all research contributions and a potential way of utilizing these research contributions beyond search computing.

Chapter 9

Conclusions

9.1 Introduction

This thesis has provided a characterization of view materialization in the context of multi domain heterogeneous search application. Specifically, the web data view materialization has been presented as a solution for technical constraints and problems implied by the unknown structure of the web data sources. To deliver this solution the web data materialization model has extended the search computing (SeCo) multi-layered model, where the search services are registered in a service repository that describes the functional (e.g., invocation end-point, input and output attributes) information of data end-points.

To deliver the web materialization as a solution to aforementioned constraints the following research goals have been tackled:

The first research goal was to solve the problem of finding a sequence of access patterns, which when executed produces a materialization output - a feasible materialization solution. We used Strongly Connected and Boundedness properties of the Petri net to derive and prove two model propositions for the analysis of the materialization reachability. The experiment results have confirmed the validity of our model and have also shown that our approach provides better performance in terms of final materialization coverage where the number of bound places is smaller in equally reachable AP layouts.

The second research goal was the optimization of the materialization process so that the most optimal sequence in terms of materialization output efficiency and quality, executes at all times. As each access pattern can be mapped to several services each differentiated by its performance and materialized data domain characteristics, the services sequence needs to be monitored

in materialization run-time by gathering and analysing predefined materialization metrics. According to materialization run-time metrics, the materialization process is optimized by switching between the available services.

The empirical study results have proved the effectiveness of the optimization as the improvement of materialization efficiency in most cases were close to three-fold when compared to the non-optimized materialization algorithms results.

9.2 Achievements and Conclusions

This section details the major achievements and describes the conclusions reached from this research.

The first main achievement is the characterization of the web materialization, its dimensions and consequent properties. The addressed dimensions were data acquisition, rank aggregation, data model, duplicate detection and data maintenance with quality assessment.

Related materialization properties included groups of Uniqueness properties, Performance properties and Service level agreement properties. Influence of one of the defined properties was observed in the case study presented in Chapter 5. The presented case study considered data coverage property in the context of data acquisition dimension. It provided new insights on the performance of classic breadth and depth first traversal algorithm in the context of materialized domain coverage.

The second main achievement is the delivery of a model and consequent methodology for materialization feasibility analysis.

In this thesis we approached feasibility analysis with two distinct objectives in mind. Firstly, the objective of the analysis was to establish which of a given set of AP combinations is capable of producing the desired materialization output for a given set of input dictionaries. In order to perform the analysis the concept of reachability was explored [Murata 1989]. A combination of access patterns was observed as a network in which input and outputs of the APs represent the nodes and the query execution instigates propagation of the values - network tokens through the network. By determining the

furthest access pattern that can be reached by a single query execution the reachability of the solution was established.

Secondly, all reachable combinations of APs were further analysed to determine the network nodes whose position shapes the network coverage - Boundedness - in terms of the number of queries this AP combination executes [Murata 1989].

Based on the presented analysis we delivered two propositions. The first proposition was based on choosing the initial access pattern for determining reachable topology of a given MPMS net. The second proposition defined a method for determining the materialization capacity (bound) of a given MPMS net determined by the number of network transient places. Two consequent algorithms were defined and implemented. The empirical study was focused on performance of the propositions. The results confirmed the validity of our propositions. We have also come to an interesting observation in regards to the bound AP attribute in respect to the final materialization outcome. The experiment results have shown that our approach provides better performance in terms of final materialization coverage when the number of bound places is smallest in equally reachable AP layouts.

The third main achievement is formalization of the materialization optimization metrics, analogue cost model and two algorithms implementing the given cost model.

The goal of optimization was to *minimize* the running time of the materialization process, while *maximizing* the produced materialization output and maintaining the right leverage between performance and the quality of the obtained materialization. Our aim was to achieve the least possible running time in the context of participating services and available input dictionaries for the prescribed materialization task.

As each access pattern can be mapped to several services, the materialization sequence needs to be monitored in materialization time by gathering and analysing predefined materialization metrics. According to materialization run-time metrics the materialization process can be adjusted by switching between available services so that the most optimal sequence in terms of materialization output efficiency and quality, executes at all times. In this situation

it is *necessary* to further differentiate between services belonging to the same access pattern.

We looked at the optimization variables and potential metrics across *service interface properties, materialized data characteristics* and properties derived from the solution feasibility analysis. Each service interface was characterized by a set of properties defining their *Uniqueness, Performance and Service Level Agreement* features.

Based on these properties we performed analysis of their influence on the materialization process as a part of several live materializations. Gathered statistics served as a basis for defining the service performance optimization metrics: WithDuplicates, ResonsePerQuery, MaxPageSize, CallsPerDay. We also used metrics derived from the domain characteristics analysis such as Frequency, Harvest Rate, Diversification and BetweennessCentrality.

We outlined and defined our optimization approach in respect to the materialization process. Further, we looked at the optimization variables and potential metrics across service interface properties, materialized data characteristics and properties derived from the solution feasibility analysis.

We proposed a cost model that employs the proposed set of optimization metrics in order to provide quantitative differentiation between the involved services and provide a base for the consequent run-time optimization algorithm. We described a set of algorithms implementing proposed optimization and illustrated their effectiveness in the empirical study.

We demonstrated the effectiveness of our optimization approach over a series of experiments. The experiments were conducted considering several materialization execution models against a medium size data set realistically mimicking 'real world' situation. The results proved the effectiveness of the approach as the improvement of materialization efficiency in most cases was close to three-fold when compared to the non-optimized materialization algorithms results.

We further demonstrated the validity of the optimization approach by performing a series of materializations of the actual web data sources that resulted in obtained materialization size close to one million tuples. The performance increase of coverage efficiencyE_C was analogous to the first set of

results. The E_C efficiency increase was present in the live experiment results although not so pronounced as in the more controlled environment of the first set of experiments.

Most of the approaches tested were able to materialize the first 50% of the given materialization task in an efficient and timely manner. However, the efficiency of the algorithms performance in the second part of the materialization process varied. The greedy algorithm performed optimally in both single queue and multi queue execution however, with decreased efficiency as the scale of the process increased in multi queue scenarios. In all tests domain structure driven query queue optimization achieved the best result once 50% or more of the materialization coverage was exceeded that is, optimal domain distribution knowledge was obtained. Given this empirical evidence it is fair to conclude that if the domain distribution of reseeding values were known a priori the query queue based optimization would deliver more significant optimization results.

The open question remains how the proposed approaches perform against a very large source with 10 - 100 million tuples sizes. The main obstacle is the availability of such sources as they were typically fronted by fairly conservative SLAs. This made our testing routine pointless as the time necessary to achieve one materialization test might stretch over an unattainable amount of time.

Lastly, we developed a **materializer module** specifically for use in Search Computing. The materializer is a software component which objective is to read arbitrary web data sources and organize data in a normalized format, suitable for data export according to an SDF definition. As data materializations are stored according to the access pattern prescribed schema, the service implementation can be automatically built by using SQL-based queries which code depends only on the service interface description.

9.3 Practical Application

A promising area for application of web data materialization is the emerging field of Computational Advertising [Broder 2008]. The central problem of Computational Advertising is to find the "best match" between a given user

in a given context and a suitable advertisement. The context could be a user entering a query in a search engine ("sponsored search"), a user reading a web page ("content match" and "display ads"), a user communicating via instant-messaging or via e-mail, a user interacting with a portable device, and many more. The information about the user can vary from scarcely detailed to practically nil. The number of potential advertisements might be in the billions. Thus, depending on the definition of "best match" this problem leads to a variety of massive optimization and search problems, with complicated constraints.

Of particular interest for web materialization application is targeted advertising, a form of personalized advertising whereby advertisers specify the features of their desired audience, either explicitly, by specifying characteristics such as demographics, location, and context, or implicitly by providing examples of their ideal audience. One of the forms of targeted advertising is a form of behavioural targeting - social targeting, where the social graph can be used to provide the data available about the user. If you are friend with many people that read Sci-Fi books, you are likely to do so as well.

There is already a developed and tested concept in twitter materialization module - Chapter 5 subsection 5.2.3 - envisaged for harvesting of the social graph - that may prove its usefulness in social targeting. Example of such usage would be gathering of audience subscribed to a specific hash tag channel or interest group. Further, aiming at such groups based on concept of homophily as friends are similar along a variety of dimensions is a long observed empirical regularity. This application may enable personal advertising opportunities to thus profiled user (potential customer) groups.

Another area where pre-fetched, materialized web data may be of interest is area of recommender systems [Schafer 1999, Broder 2008]. Main objective of recommender systems is to recommend items based on explicit (rating) or implicit (page visits, ad clicks). Many web sites offer virtually unrestricted access to information such as 'customers who bought this items also bought' in case of Amazon.com. Web materialization may be used to simply harvest user ratings of particular items or of groups of items such as movies, books or restaurants in order to deliver personalized and previous usage based rec-

ommendations to the next 'page visitor' such as Yahoo.com or google.com shopping or Netflix.com.

Alternatively, materialization may find its application in gathering information from domain specific search engines such as booking.com or tripadvisor .com. Thus obtained data is further integrated in federated search engines with aim to deliver a 'one-shop-stop' user experience [Ceppi 2012].

9.4 Final Remarks

The introduction to this thesis highlighted the need of the heterogeneous multi domain search application for reliable and readily available data sources. With this in mind we have approached the notion of view materialization and have transposed it into a vehicle for reaching to and sourcing from the vast bed of knowledge hidden within the deep web.

We believe that this thesis has provided comprehensive research and a way forward in this area.

Moreover, the extensions proposed in the previous Chapter are but a few of the improvements required to enhance performance in this new and challenging area of research. Algorithmic advances in query reseeding, data surfacing and query processing will enhance performance further, but so will the provision of new, more informative ontologies and domain exploration techniques.

Appendix A

Appendix

A.1 Optimization Result Sets

In this appendix we present all the optimization procedure result runs with average standard deviation highlighted in light and dark grey rows respectively.

	SSQ RANDOM							
Run no.	10%	20%	30%	40%	50%	60%	70%	80%
1	631	1541	2180	2777	4754	5259	7079	7906
2	558	1234	2404	2714	4033	5130	7000	8113
3	690	1643	2482	2696	4279	4312	5527	6493
4	590	1566	1882	2725	3642	5294	5611	7228
5	560	1577	2528	3217	3608	4950	5733	8218
6	552	1373	2137	3190	4844	4967	6248	7951
7	689	1605	2534	3048	3624	5648	5982	7198
8	712	1628	1870	2761	3945	5678	5958	5936
9	533	1488	2364	3075	4692	4829	7340	7849
10	534	1420	2512	3056	4737	4724	6535	5760
11	759	1502	2214	3298	3994	6118	6040	6070
12	610	1564	1984	3381	4139	5812	5887	6418
13	560	1643	1881	3735	4833	5748	6648	6424
14	687	1451	2232	3219	4190	4558	5976	6629
15	554	1459	2675	2765	3437	6107	6121	7524
16	608	1532	2488	2846	4024	5907	5577	6457
17	726	1392	2017	2843	4586	5595	7564	7435
18	686	1595	1975	3685	3855	4429	6471	7210
19	634	1290	2493	3330	3771	5389	6057	7434
20	704	1286	2190	3296	4587	5606	7824	8283
21	679	1447	2456	3365	3919	5413	6149	6721
22	622	1368	1875	2895	3445	5007	7041	7005
23	568	1187	2042	2918	4375	5349	7522	6539
24	564	1463	2348	3072	3816	4782	6358	7016
25	696	1408	2463	3257	3735	5067	7096	7137
26	755	1174	2229	2844	3754	4445	6327	7052
27	717	1280	1957	2916	4514	4940	7529	7922
28	754	1499	1875	2833	4309	5374	7125	6237
29	714	1633	2706	3224	3495	5189	7818	6190
30	525	1188	2391	3133	3503	5150	7532	6535
AVG	639	1448	2246	3070	4081	5226	6589	7030
STDEV	76	145	262	278	452	487	721	721

Figure A.1: Serial single queue random runs.

	SSQ PARALLEL									
Run no.	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	416	980	1297	2131	2886	3289	3801	4974	6027	8120
2	352	971	1240	2204	2629	3636	4258	6453	5992	8132
3	369	723	1533	2192	2949	3449	5246	5916	7416	7399
4	323	924	1243	2164	3112	3748	3896	4629	7008	6673
5	311	983	1169	1986	2968	3195	5194	5057	6777	6454
6	368	704	1472	1729	2356	3404	4201	5009	5375	8708
7	402	705	1225	1783	2547	4037	4843	5713	6723	8261
8	391	981	1372	1707	2806	3515	3898	5075	5989	6996
9	414	808	1184	2419	2300	3689	5011	5065	5844	6457
10	316	797	1360	1727	2771	3694	3671	4534	7589	6273
11	386	706	1352	2146	3098	3301	4844	6211	5310	7393
12	407	778	1350	1935	2938	3576	3966	4783	6057	8146
13	310	875	1439	1889	3099	3682	4611	5387	6018	8791
14	426	749	1421	1875	2850	3145	4299	5813	5617	7934
15	358	877	1246	2125	2632	3650	4363	6214	6010	8720
16	324	782	1382	2217	3009	2902	5099	4803	7004	7658
17	381	904	1162	2412	2305	4001	3709	5448	6795	8182
18	380	819	1303	2025	2674	3332	4369	5472	7356	7099
19	301	786	1408	1895	2766	2896	5147	5990	6211	7920
20	316	934	1478	2130	3001	3358	5261	5293	5590	7612
21	406	718	1394	2222	2623	3411	3789	4621	7088	6403
22	367	876	1540	2323	2706	3812	4255	4674	5550	7891
23	332	720	1585	2026	2588	3099	4264	5524	6050	8437
24	425	905	1246	1834	2379	3631	4247	5327	7296	6988
25	332	983	1347	2306	2619	3634	4014	6125	7050	8172
26	392	832	1208	1699	2730	3236	4943	5204	5334	7950
27	334	795	1397	1929	3081	3378	3820	4586	7108	8314
28	368	900	1125	1867	2649	3968	4362	5714	6939	8121
29	349	838	1405	2178	2910	3389	4064	5261	6232	6499
30	382	858	1536	2218	3120	3864	4463	5024	5698	7653
AVG	365	840	1347	2043	2770	3497	4397	5330	6368	7645
STDEV	38	93	124	212	246	299	506	544	700	758

Figure A.2: Serial single queue parallel runs.

	SSQ OPT									
Run no.	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	219	654	1060	1589	2218	2882	3230	5273	5320	6391
2	233	512	1271	1885	2235	2989	4145	4304	6328	6900
3	279	558	1090	1376	2114	3059	3846	4059	5108	7660
4	263	631	1329	1885	1928	3014	3644	5352	6745	6921
5	244	546	1309	1704	2435	2847	3610	4622	6551	6520
6	259	504	1193	1871	2396	3116	3668	4663	5922	8260
7	221	509	1205	1556	2654	3075	4455	5058	5174	6263
8	247	715	1254	1847	2142	2758	3585	5071	5960	7107
9	299	622	1161	1466	2202	2597	3181	5384	6964	6582
10	213	516	1264	1545	2553	2954	4410	4833	6130	6750
11	246	650	1160	1779	2549	2702	3768	4676	5867	7816
12	238	589	1191	1810	2391	2673	4273	4723	6647	6134
13	249	597	945	1581	1995	2590	3559	5029	5290	8066
14	278	589	1095	1425	2380	3152	4192	5300	5417	6334
15	222	509	1259	1541	2161	2983	4065	4468	5646	7644
16	272	715	1114	1856	2049	2759	3588	4114	6615	7123
17	284	617	1186	1757	2686	2467	3488	5212	5925	7986
18	262	629	1171	1458	2275	3465	3304	4867	5918	7073
19	293	562	1083	1631	2672	2511	3918	4794	5083	8717
20	278	685	1195	1717	2082	3021	3171	4037	7041	7813
21	277	666	968	1650	2418	3477	3655	5435	5974	7863
22	229	546	1201	1749	2046	2866	3504	5150	5317	7122
23	256	631	984	1457	2368	3352	3648	5101	5318	8066
24	287	596	1182	1738	2287	3045	3523	4466	6327	7095
25	228	517	1188	1544	2046	3153	3868	5169	5398	7981
26	263	620	969	1699	2257	3143	4084	5242	5559	6500
27	288	684	1291	1674	2259	3086	3524	4783	5410	8236
28	308	535	964	1461	1920	3134	4083	4259	5102	6285
29	268	553	1056	1860	1912	3290	3742	5171	6613	8270
30	291	616	953	1690	2221	2867	4095	4343	6972	6551
AVG	260	596	1143	1660	2262	2968	3761	4832	5921	7268
STDEV	26	64	114	154	222	261	350	423	629	749

Figure A.3: Serial single queue full optimization runs.

	SSQ OPT GREEDY									
Run no.	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	265	525	956	1214	1577	2401	2813	3450	3772	6080
2	237	604	885	1182	1635	2102	2549	3583	4656	4944
3	256	569	740	1341	1463	2255	2972	3886	4356	4563
4	240	461	853	1438	1564	2227	2746	3982	3979	5315
5	264	611	715	1285	1678	2225	2916	3014	4272	5590
6	233	560	712	1177	1874	1905	3164	4121	4705	4948
7	294	576	743	1088	1877	2104	2683	3447	5441	5789
8	257	639	1015	1195	1573	1998	2513	2919	5133	4925
9	272	595	816	1261	1878	2620	2377	3967	3888	6353
10	207	471	882	1501	1781	2391	3247	4050	4118	6504
11	261	456	712	1155	1630	2492	2875	3881	4497	5744
12	264	605	839	1195	1662	2407	3141	2894	4244	6185
13	214	459	980	1414	1688	2164	2653	3068	4664	4765
14	227	462	754	1091	1829	2299	2866	2894	4695	6326
15	283	542	913	1230	1727	2534	2626	3497	5017	4610
16	265	453	998	1391	1858	1984	2783	3623	4310	5583
17	288	497	960	1249	2003	2125	2795	3014	4449	4882
18	257	584	872	1200	1918	2221	2954	3625	5386	5064
19	288	516	999	1187	1669	2347	2656	3807	5200	6436
20	282	499	864	1281	1855	2527	2835	3794	4029	5273
21	240	543	937	1358	1762	1848	2436	3023	4135	4878
22	252	557	927	1428	1633	2169	2491	3269	4440	5510
23	225	503	790	1298	1466	1882	3174	2920	4570	4971
24	254	602	882	1182	1691	2323	2637	2924	4271	4613
25	256	617	984	1249	1548	2265	2604	3476	4434	4910
26	258	590	739	1256	1711	2176	3004	3222	4471	4970
27	226	546	902	1428	1455	2528	2318	4184	5322	4996
28	245	510	962	1473	1992	1898	2576	3436	3849	6515
29	222	552	870	1127	1531	2046	2757	4048	4269	5778
30	238	637	766	1306	1670	2147	3194	3516	4886	5567
AVG	252	545	866	1273	1707	2220	2779	3484	4515	5420
STDEV	23	58	97	114	154	211	252	423	461	625

Figure A.4: Serial single queue full greedy optimization runs.

	SSQ SERVICE OPT									
Run no.	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	288	741	1068	1287	2479	2970	3852	4393	6043	6321
2	243	656	1123	1567	2417	2994	3327	4561	5204	6503
3	270	626	962	1809	2149	3395	4003	4938	6286	6101
4	265	713	1296	1494	1862	2414	3858	5025	5578	6486
5	261	660	1097	1413	1895	3421	3970	4938	6247	7605
6	226	673	1221	1596	2283	3394	3274	5452	5707	7613
7	293	571	1201	1351	1796	3219	3260	4548	6361	7839
8	255	572	918	1710	2251	2752	3568	4976	5843	7658
9	283	584	948	1554	1893	2816	4286	4393	6641	7623
10	249	704	1178	1574	1979	2381	3357	4886	5629	6129
11	225	741	1063	1685	2111	3210	4167	4963	5243	8006
12	252	738	1047	1265	1960	3244	3856	4100	5012	6671
13	254	738	1291	1726	2189	3165	3345	4291	6937	8414
14	248	602	1073	1584	2242	3140	3524	5423	5635	6512
15	314	722	1108	1736	2090	3286	4078	4855	7026	8059
16	321	593	916	1794	2045	3023	3947	5447	5343	6962
17	310	625	976	1416	2387	3122	3365	4516	6863	6807
18	224	745	1248	1441	2452	2967	4333	4389	6755	7171
19	261	523	1206	1519	2404	2853	4632	4870	6868	8462
20	312	533	1092	1365	2284	2598	3631	5322	5261	7554
21	225	519	901	1671	2211	2809	3504	5747	6856	8441
22	257	681	1055	1444	2383	2828	4627	4259	6616	8362
23	284	517	1173	1329	1951	2532	3581	5428	6449	7257
24	294	691	1146	1798	2247	3339	4686	5327	6259	6727
25	306	646	1081	1251	1961	2933	4341	5539	5154	6174
26	299	607	1189	1256	1818	3016	3887	5411	5463	6678
27	279	524	1035	1474	2279	3134	3793	5556	5462	7137
28	229	566	1062	1582	1953	3444	4437	4352	7094	6637
29	285	608	1229	1562	2067	3376	3963	5363	6681	7433
30	242	723	1235	1701	2321	3013	3810	5310	6221	7739
AVG	268	638	1105	1532	2145	3026	3875	4953	6091	7236
STDEV	30	77	113	171	204	295	425	475	658	740

Figure A.5: Serial single queue service optimization runs.

	SSQ SERVICE OPT GREEDY									
Run no.	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	218	450	830	1149	1485	2414	3257	4603	5623	5955
2	264	482	869	1438	1611	2304	3072	3930	4365	4925
3	258	492	741	1058	1414	2124	2462	4302	4503	5532
4	217	556	778	1488	1793	2515	3314	4175	5842	5964
5	256	491	872	1229	1746	2281	3080	4205	5360	5538
6	225	465	880	1271	1805	2213	2822	4544	5712	5488
7	217	617	950	1161	1537	2322	2703	4142	4993	6878
8	196	566	749	1068	1483	2379	2639	3568	4640	6670
9	210	534	911	1267	1466	2450	3159	3868	5569	5192
10	255	550	1013	1352	1501	2228	3186	4212	4378	6442
11	225	617	784	1509	1842	2000	2650	3950	5984	6216
12	200	637	898	1283	1796	2438	2807	3598	4789	5644
13	240	500	1027	1211	1944	2064	3371	4570	4954	6127
14	274	531	793	1366	1537	2089	2729	3453	4956	5390
15	260	577	949	1145	1920	2179	3042	3767	4530	6059
16	223	499	966	1251	1763	2186	3359	4360	5447	5376
17	231	511	940	1302	1961	2153	2882	3754	4977	6130
18	223	584	955	1094	1596	2304	2889	3807	5817	5022
19	253	456	982	1389	1822	2406	2560	3291	4468	6100
20	238	629	934	1333	1958	2509	3297	3938	5343	6832
21	250	527	947	1255	1441	2503	2984	4531	4677	5811
22	206	478	768	1145	1578	2270	2451	3959	4250	6804
23	245	492	832	1088	1600	2141	2946	4682	4332	5834
24	244	558	989	1346	1698	1933	3086	4303	4378	5086
25	243	570	937	1205	1703	1952	2485	4286	5791	6256
26	224	585	804	1108	1620	2130	2883	4284	4236	6807
27	253	482	851	1258	1646	1829	2587	3510	4618	5440
28	235	446	804	1126	1880	2342	3123	3435	5094	6150
29	208	504	944	1077	1405	2630	3063	4647	5214	5218
30	202	502	878	1258	1563	2587	3278	3554	4598	5809
AVG	233	530	886	1241	1670	2263	2939	4041	4981	5890
STDEV	21	55	82	125	173	201	284	407	551	570

Figure A.6: Serial single queue service greedy optimization runs.

	SMQ RANDOM									
Run no.	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	862	2404	3454	4762	6927	6230	7558	9918	9535	10269
2	943	1882	3461	5314	6731	7963	6838	8906	9478	10670
3	766	2146	2956	5635	5956	6771	7970	7890	9969	11927
4	764	2361	3059	4581	6709	7308	9474	7441	10061	11668
5	884	2107	3337	5171	6610	6261	7170	8591	11499	12441
6	760	2100	3244	3999	5821	8168	7595	7839	11976	13850
7	936	2072	4006	4772	5796	7796	6879	8297	8441	11961
8	862	2298	3175	5029	5486	8542	9478	9521	10261	12153
9	793	2387	3463	5662	5697	6947	7006	8991	9501	11526
10	940	1841	2958	4399	5997	6785	7037	8875	11534	12885
11	820	1817	3618	4467	6176	6971	8086	7867	8874	12453
12	769	1805	3873	4217	6050	7432	8973	8996	11836	11143
13	883	1851	3872	5341	7041	7485	7496	10093	11093	12593
14	1023	2212	3498	4672	5846	8364	9125	7889	11306	12596
15	973	1972	3800	5618	7314	7553	7471	8827	8912	11121
16	953	2321	2779	4187	5819	8006	8400	8527	8555	12741
17	882	1821	3613	5025	5447	6733	8010	8486	9830	10344
18	1025	2456	3374	4602	5534	6807	7835	9552	8920	12508
19	1010	1797	2972	4504	5578	6874	7956	9529	8987	10536
20	923	2107	3089	4650	7132	7086	8014	7417	11597	13666
21	1000	2051	3583	5549	5464	6354	9385	9562	11329	13717
22	1072	2155	3181	4706	5705	6818	8661	7874	9989	12947
23	990	1846	3878	4365	5672	7596	6913	9190	12005	14287
24	882	2118	3391	4085	6356	8449	6896	9077	8419	13298
25	930	1975	3610	4532	6938	8690	8670	9970	8936	11909
26	981	1718	3887	5205	6164	7640	7475	9185	9878	10343
27	1009	1994	3389	5438	6858	6914	9361	9712	9755	10964
28	1026	2245	3248	4299	6059	6398	9419	9339	10194	10671
29	906	1835	3864	5091	6112	6076	8457	8957	8618	13835
30	1015	1865	2819	5119	6253	7043	6906	9335	10009	11232
AVG	919	2052	3415	4833	6175	7269	8017	8855	10043	12075
STDEV	90	214	347	494	557	733	902	762	1155	1183

Figure A.7: Serial multi queue random runs.

	SMQ PARALLEL									
Run no.	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	571	1130	1828	2663	4485	5497	7388	9692	10129	13075
2	459	1379	1860	2866	4198	5350	7055	9195	11189	14045
3	464	1126	1792	3502	4137	5573	6531	8020	11104	11086
4	451	1322	2246	3332	4781	5006	8141	10048	11819	12548
5	491	1058	2287	3327	4692	5463	5875	7643	11561	10075
6	591	1236	1765	3467	4235	6083	5954	9328	11493	11586
7	596	1130	1841	3454	4998	4704	6592	9025	10870	12371
8	588	1304	1874	3199	4911	5996	7934	8007	11409	13125
9	462	1329	1881	3635	4587	6319	7690	7640	11426	12758
10	624	1331	2127	2993	4286	6658	7427	7701	11452	11091
11	447	1311	1821	2646	4776	4709	7858	8443	11483	13050
12	582	1270	1959	2778	4290	5482	7648	9469	9870	12415
13	540	1365	1890	2680	4658	5562	7032	8626	12072	10830
14	553	1064	2238	2726	4073	4995	7484	10602	11199	13451
15	469	1315	1909	2987	4667	5450	6284	8400	9256	13103
16	447	1060	2194	3064	4799	6022	6700	9823	8576	11859
17	567	1260	1875	3665	4589	6092	8101	9676	11049	10534
18	604	1215	2195	2833	4130	6298	8188	8314	8489	13617
19	496	1274	2306	3087	4449	5349	8453	10033	9758	13897
20	581	1067	1938	2902	4164	6041	5839	8310	9851	13595
21	592	1246	1835	3166	4230	6316	6344	8461	8768	14441
22	551	1119	2124	3164	3847	6137	7545	9017	10555	13339
23	500	1312	2284	3346	4660	6515	8220	8690	8910	13596
24	450	1291	1971	2667	4121	4888	7442	7795	11093	10187
25	512	1183	2208	3527	4432	5568	8331	9444	9251	12620
26	516	1108	1807	3328	3818	5321	7882	7850	10162	13784
27	588	1253	2294	3560	3879	6622	6474	8572	9124	13791
28	445	1169	2131	3063	5035	5091	7402	8836	10959	11821
29	448	1162	2217	3201	3609	5058	6950	8721	11815	12268
30	512	1199	1769	3181	4423	5953	7412	7561	10642	10658
AVG	523	1220	2016	3134	4399	5671	7273	8765	10511	12487
STDEV	60	99	191	313	366	570	769	827	1080	1248

Figure A.8: Serial multi queue parallel runs.

	SMQ OPT									
Run no.	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	368	1071	1625	2719	3492	4833	5947	8612	10428	14212
2	351	813	1885	2489	3555	4680	7363	8895	10190	10616
3	432	976	1437	2965	4055	4331	7572	10306	8915	10145
4	393	832	1740	2437	3388	5584	6353	8319	9243	10745
5	364	793	1374	3004	3761	4635	5833	9045	10938	13263
6	407	942	1782	2906	3248	5190	7653	10274	8736	11355
7	349	950	1508	2726	3361	5563	8027	8643	8570	12835
8	386	753	1835	2844	4271	4542	5740	8342	11586	13661
9	343	1001	1453	2242	3590	4735	6834	7290	8662	11336
10	392	1014	1537	2548	4026	4573	6604	9763	8605	11729
11	310	935	1496	2445	4052	4727	7472	9591	11181	10180
12	324	938	1361	2506	4420	5168	7800	9076	10498	12619
13	436	903	1842	2556	4028	5120	5841	9025	11678	13954
14	388	1001	1665	2928	4073	5407	7013	9384	8850	14037
15	433	901	1765	2925	3583	4784	7272	8241	11893	14237
16	319	869	1699	2732	3699	4462	7733	7766	11114	13622
17	308	919	1454	2241	3457	4277	7854	7198	12071	12749
18	323	960	1410	2429	3979	5170	7138	7751	9800	11582
19	349	880	1658	2109	4396	4754	7719	7659	8530	10165
20	352	958	1802	2213	3690	4320	5528	9466	10075	10746
21	438	941	1846	2740	3909	5022	8021	7524	9544	13075
22	442	1013	1833	2631	3608	5004	6845	8022	11201	12321
23	403	792	1841	2425	4393	4975	7580	8827	11236	13776
24	418	1071	1773	2441	4061	5493	7822	7461	8448	13140
25	333	932	1842	2429	3513	4635	5917	7203	10940	11201
26	334	953	1874	2900	3837	5621	6756	7490	12121	11357
27	369	999	1431	2291	3792	5503	6028	9530	10469	13129
28	349	845	1729	2135	3407	5262	6857	10139	10679	11182
29	381	859	1856	2419	3977	4403	6905	9315	9681	14339
30	365	892	1405	2197	4034	5895	7227	8039	8532	12696
AVG	372	924	1659	2552	3822	4956	6975	8607	10147	12333
STDEV	41	80	181	269	329	445	769	955	1219	1366

Figure A.9: Serial multi queue full optimization runs.

	SMQ OPT GREEDY									
Run no.	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	293	757	1427	1542	2719	2847	3619	5023	7675	12068
2	305	682	1132	1994	2363	3619	4474	5702	10332	11858
3	280	768	1229	1927	2318	2869	4273	5204	9428	11489
4	344	645	1320	1783	3008	2863	4015	6665	7333	11606
5	285	756	1320	1863	2895	3550	4103	6126	7947	12559
6	320	660	1122	1838	2935	4005	4790	5876	7830	13480
7	374	884	1473	1528	2480	3239	3706	5797	8833	11192
8	311	707	1502	1988	2826	3857	4069	7071	10321	11949
9	387	840	1224	1659	2836	3283	4461	6038	9114	10906
10	341	675	1036	1537	2129	3699	4933	6413	8583	12206
11	341	781	1195	1826	2600	3823	4581	5701	8670	12553
12	299	898	1413	1747	2307	3855	4268	5599	7251	12900
13	379	746	1083	1701	2520	2851	3907	5797	9880	12002
14	401	836	1446	1673	2298	3593	4979	5678	8702	11639
15	344	667	1038	1835	2534	3391	4178	6167	7109	11227
16	352	737	1117	1946	2583	3828	4665	6171	9502	13623
17	286	844	1252	1835	2474	3275	4770	6589	9022	13397
18	387	682	1137	1574	2871	3607	4479	6261	9153	13044
19	284	899	1277	1975	2700	3290	4789	6723	8926	12161
20	354	701	1292	1585	2502	3015	3459	5050	9006	14311
21	394	886	1171	1562	2961	3483	4132	5903	9117	13336
22	301	635	1452	1545	3022	2842	3705	6427	9612	11551
23	298	782	1404	1837	2199	3579	3466	6221	7556	10860
24	311	768	1351	1779	3009	2828	4921	7105	7159	12968
25	313	784	1180	1661	2189	3616	3440	5787	8552	11926
26	284	641	1099	2182	2483	3448	3433	6001	9715	10570
27	302	794	1461	2058	2254	3163	4659	5892	9260	12369
28	298	722	1357	1795	2847	2971	3469	5084	7721	12224
29	309	678	1132	1943	2560	3188	3491	6154	7325	12535
30	289	740	1105	1674	2962	3709	3877	6675	10260	13136
AVG	326	753	1258	1780	2613	3373	4170	6030	8696	12255
STDEV	38	80	144	175	281	365	520	545	995	906

Figure A.10: Serial multi queue full greedy optimization runs.

	SMQ SERV OPT									
Run no.	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	357	821	1885	2414	4264	6758	8008	9837	12467	11992
2	337	887	1497	2768	3779	6443	6858	10452	9991	12098
3	412	1025	1657	2488	4085	6155	7577	9203	12641	12527
4	387	994	1432	2227	3318	6336	8843	9557	11491	11685
5	387	929	1399	2124	3353	6293	6653	9747	8787	12507
6	395	939	1815	2878	4643	5612	8097	10455	11063	10233
7	378	820	1592	2372	4559	6494	6509	10084	9157	13790
8	361	817	1792	2534	4310	6492	8508	9141	11494	10311
9	336	1028	1983	2652	4344	5205	8590	10913	10276	13579
10	403	946	1464	2935	4205	6036	7385	8325	11387	11253
11	311	1049	1591	2710	3493	6713	9454	10191	9856	10704
12	372	852	1753	2686	4370	6057	7798	11066	9102	11543
13	358	873	1721	2772	4564	5634	9403	9477	11183	11056
14	400	969	1567	2401	3711	6091	8708	10031	10910	11363
15	338	864	1498	2762	3688	4800	7962	10724	10021	10938
16	346	747	1641	2220	3506	4840	7437	9327	11281	14279
17	359	871	1846	2625	3389	5060	7563	9207	11828	13644
18	363	1074	1788	2236	4340	5008	6810	10371	9809	14315
19	412	789	1459	2287	3373	5539	7661	10756	10605	12362
20	381	850	1747	2393	3730	5759	7482	8458	11486	10095
21	371	1031	1972	2601	3361	5805	7064	9741	11628	11085
22	403	978	1639	2424	3785	5829	8748	9133	10053	12340
23	414	901	1868	2348	4059	6428	6862	10098	11684	11349
24	389	932	1770	2662	3916	5621	8241	9511	9674	10166
25	367	977	1889	2147	3751	5613	6844	8048	9745	12016
26	402	967	1946	3014	3414	6813	6942	7994	9963	11996
27	344	759	1877	2254	3908	6490	8572	8635	11208	11131
28	420	973	1838	2526	4158	5595	7863	8170	11291	13742
29	400	857	1590	2759	3598	4988	8742	9030	11908	10546
30	416	1019	1688	2727	4534	5463	8375	8972	9070	10773
AVG	377	918	1707	2532	3917	5866	7852	9555	10702	11847
STDEV	29	89	171	244	425	594	826	872	1050	1255

Figure A.11: Serial multi queue service optimization runs.

	SMQ SERV OPT GREEDY									
Run no.	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	327	659	1197	1542	2499	3725	4428	6061	10049	12942
2	326	868	1237	1946	3131	3513	5155	6107	7973	10866
3	340	809	1267	2169	2299	4277	4029	6355	8962	11934
4	386	795	1270	1981	2423	4118	5467	6424	7952	13615
5	324	837	1445	2022	3174	3961	4134	5230	8710	12177
6	302	767	1182	1879	3120	3453	5056	7292	8380	13587
7	324	771	1353	1842	2640	3185	4212	5584	9614	11316
8	346	718	1165	2176	2371	3968	4095	5434	8402	13492
9	360	891	1437	2112	2508	4252	4747	5530	10273	11893
10	337	731	1210	1989	2739	4237	4253	5364	8809	11865
11	403	732	1103	1894	2461	3131	4485	6050	8312	10885
12	312	749	1371	1858	2916	3206	4864	6056	8275	10892
13	294	761	1453	1833	3095	3827	4314	5701	10262	12482
14	331	822	1153	2207	2810	3722	4583	6927	10145	13063
15	385	665	1378	1729	2240	3493	4049	6218	9779	11645
16	336	776	1285	1768	2614	3485	5035	5438	7595	13386
17	329	873	1184	2046	2733	3595	4281	7140	7790	11897
18	337	777	1148	2140	2481	3586	4164	5465	8508	10586
19	301	620	1137	2064	2546	3658	4494	5319	8647	9961
20	285	632	1213	2231	2328	3780	4958	5845	8268	13375
21	285	868	1194	2095	2392	3814	5540	6766	10431	11425
22	295	720	1196	1646	3039	3061	4642	5278	7837	12505
23	301	719	1142	1903	2700	3951	4102	7297	8632	11947
24	370	734	1067	1939	2735	3496	4070	6911	9393	11170
25	322	666	1106	2202	2678	3016	4954	7337	9122	10562
26	326	757	1408	2060	2661	4083	4174	6431	9650	14099
27	366	718	1395	2051	2704	3805	4142	6822	8630	11136
28	328	872	1132	2061	2554	3464	3843	6984	9157	10892
29	327	633	1288	1864	2450	3997	3921	5662	9859	11640
30	321	710	1429	1593	3194	4140	4806	5943	8384	11437
AVG	331	755	1252	1961	2675	3700	4500	6166	8927	11956
STDEV	30	77	117	183	278	362	462	685	835	1085

Figure A.12: Serial multi queue service greedy optimization runs.

Bibliography

[Aalst 1999] Wvd Aalst. *Interorganizational workflows: An approach based on message sequence charts and petri nets.* Systems Analysis-Modelling-Simulation, vol. 34, no. 3, pages 335–367, 1999. (Cited on page 22.)

[Adali 1996] Sibel Adali, K Selçuk Candan, Yannis Papakonstantinou and VS Subrahmanian. *Query caching and optimization in distributed mediator systems.* In ACM SIGMOD Record, volume 25, pages 137–146. ACM, 1996. (Cited on page 3.)

[Adler 2001] Micah Adler and Michael Mitzenmacher. *Towards compressing web graphs.* In Data Compression Conference, 2001. Proceedings. DCC 2001., pages 203–212. IEEE, 2001. (Cited on pages 27 and 28.)

[Agrawal 2000] Sanjay Agrawal, Surajit Chaudhuri and Vivek R Narasayya. *Automated Selection of Materialized Views and Indexes in SQL Databases.* In VLDB, volume 2000, pages 496–505, 2000. (Cited on page 16.)

[Altinel 2007] Mehmet Altinel, Paul Brown, Susan Cline, Rajesh Kartha, Eric Louie, Volker Markl, Louis Mau, Yip-Hing Ng, David Simmen and Ashutosh Singh. *Damia: a data mashup fabric for intranet applications.* In Proceedings of the 33rd international conference on Very large data bases, pages 1370–1373. VLDB Endowment, 2007. (Cited on page 21.)

[Amiri 2003] Khalil Amiri, Sanghyun Park, Renu Tewari and Sriram Padmanabhan. *DBProxy: A dynamic data cache for Web applications.* In 2013 IEEE 29th International Conference on Data Engineering (ICDE), pages 821–821. IEEE Computer Society, 2003. (Cited on page 16.)

[Aouiche 2009] Kamel Aouiche and Jérôme Darmont. *Data mining-based materialized view and index selection in data warehouses.* Journal of Intelligent Information Systems, vol. 33, no. 1, pages 65–93, 2009. (Cited on page 16.)

[Atributor 2014] Atributor. *Atributor.com*, 2014. Offline; accessed 30-September-2014. (Cited on page 29.)

[AutoAdmin 2014] AutoAdmin. *AutoAdmin*, 2014. Online; accessed August-2015. (Cited on page 6.)

[Baeza-Yate 2007] Ricardo Baeza-Yate, Flavio Junqueira, Vassilis Plachouras and Hans Friedrich Witschel. *Admission policies for caches of search engine results*. In String Processing and Information Retrieval, pages 74–85. Springer, 2007. (Cited on page 26.)

[Baeza-Yates 2003] Ricardo Baeza-Yates and Felipe Saint-Jean. *A three level search engine index based in query log distribution*. In String Processing and Information Retrieval, pages 56–65. Springer, 2003. (Cited on page 26.)

[Baeza-Yates 2007] Ricardo Baeza-Yates, Aristides Gionis, Flavio Junqueira, Vanessa Murdock, Vassilis Plachouras and Fabrizio Silvestri. *The impact of caching on search engines*. In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pages 183–190. ACM, 2007. (Cited on page 26.)

[Bailey 2005] James Bailey, Francois Bry, Tim Furche and Sebastian Schaffert. *Web and semantic web query languages: A survey*. In Proceedings of the First international conference on Reasoning Web, pages 35–133. Springer-Verlag, 2005. (Cited on page 19.)

[Bar-Yossef 2008] Ziv Bar-Yossef and Maxim Gurevich. *Random sampling from a search engine's index*. Journal of the ACM (JACM), vol. 55, no. 5, page 24, 2008. (Cited on page 27.)

[Barbosa 2004] Luciano Barbosa and Juliana Freire. *Siphoning Hidden-Web Data through Keyword-Based Interfaces*. In SBBD, pages 309–321, 2004. (Cited on page 31.)

[Baru 1999] Chaitan Baru, Amarnath Gupta, Bertram Ludäscher, Richard Marciano, Yannis Papakonstantinou, Pavel Velikhov and Vincent Chu.

XML-based information mediation with MIX. In ACM SIGMOD Record, volume 28, pages 597–599. ACM, 1999. (Cited on page 17.)

[Ben-Abdallah 1997] Hanêne Ben-Abdallah and Stefan Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. Springer, 1997. (Cited on page 24.)

[Berger 2003] Sacha Berger, François Bry and Sebastian Schaffert. *A visual language for Web querying and reasoning.* In Principles and Practice of Semantic Web Reasoning, pages 99–112. Springer, 2003. (Cited on page 19.)

[Berger 2004] Sacha Berger, François Bry, Oliver Bolzer, Tim Furche, Sebastian Schaffert and Christoph Wieser. *Xcerpt and visxcerpt: Twin query languages for the semantic web.* In Proc. Int. Semantic Web Conf, volume 11, page I4, 2004. (Cited on page 19.)

[Bharat 1998] Krishna Bharat and Andrei Broder. *A technique for measuring the relative size and overlap of public web search engines.* Computer Networks and ISDN Systems, vol. 30, no. 1, pages 379–388, 1998. (Cited on page 27.)

[Bhogal 2007] Jagdev Bhogal, Andy Macfarlane and Peter Smith. *A review of ontology based query expansion.* Information processing & management, vol. 43, no. 4, pages 866–886, 2007. (Cited on page 188.)

[Bianchini 2006] Devis Bianchini, Valeria De Antonellis, Barbara Pernici and Pierluigi Plebani. *Ontology-based methodology for e-service discovery.* Information Systems, vol. 31, no. 4, pages 361–380, 2006. (Cited on page 21.)

[Bozzon 2010] Alessandro Bozzon, Marco Brambilla, Stefano Ceri and Piero Fraternali. *Liquid query: multi-domain exploratory search on the web.* In WWW '10: Proceedings of the 19th international conference on World wide web, pages 161–170, New York, NY, USA, 2010. ACM. (Cited on page 37.)

[Bozzon 2011a] Alessandro Bozzon, Daniele Braga, Marco Brambilla, Stefano Ceri, Francesco Corcoglioniti, Piero Fraternali and Salvatore Vadacca. *Search computing: multi-domain search on ranked data*. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, pages 1267–1270. ACM, 2011. (Cited on pages 37 and 104.)

[Bozzon 2011b] Alessandro Bozzon, Marco Brambilla, Stefano Ceri and Piero Fraternali. Information exploration in search computing. Springer, 2011. (Cited on pages 37 and 169.)

[Bozzon 2012] Alessandro Bozzon, Stefano Ceri and Srđan Zagorac. *Materialization of web data sources*. In Search Computing, pages 68–81. Springer, 2012. (Cited on page 69.)

[Braga 2010a] Daniele Braga, Stefano Ceri, Francesco Corcoglioniti and Michael Grossniklaus. Panta rhei: flexible execution engine for search computing queries. Springer, 2010. (Cited on page 59.)

[Braga 2010b] Daniele Braga, Stefano Ceri and Michael Grossniklaus. Join methods and query optimization. Springer, 2010. (Cited on page 59.)

[Braga 2011] Daniele Braga, Michael Grossniklaus, Francesco Corcoglioniti and Salvatore Vadacca. Efficient computation of search computing queries. Springer, 2011. (Cited on pages 20, 37, 38, 40 and 42.)

[Brambilla 2011] Marco Brambilla, Alessandro Campi, Stefano Ceri and Silvia Quarteroni. Semantic resource framework. Springer, 2011. (Cited on pages 6, 37, 38, 54 and 55.)

[Breiman 1984] Leo Breiman, Jerome Friedman, Charles J Stone and Richard A Olshen. Classification and regression trees. CRC press, 1984. (Cited on page 49.)

[Broder 2008] Andrei Z Broder. *Computational advertising and recommender systems*. In Proceedings of the 2008 ACM conference on Recommender systems, pages 1–2. ACM, 2008. (Cited on pages 201 and 202.)

[Cafarella 2009] Michael J. Cafarella, Jayant Madhavan and Alon Halevy. *Web-scale extraction of structured data*. SIGMOD Rec., vol. 37, pages 55–61, March 2009. (Cited on pages 3, 29 and 30.)

[Calı 2009] Andrea Calı, Diego Calvanese and Davide Martinenghi. *Dynamic query optimization under access limitations and dependencies*. Journal of Universal Computer Science, vol. 15, no. 21, pages 33–62, 2009. (Cited on page 32.)

[Callan 1999] Jamie Callan, Margaret Connell and Aiqun Du. *Automatic discovery of language models for text databases*. In ACM SIGMOD Record, volume 28, pages 479–490. ACM, 1999. (Cited on page 28.)

[Callan 2000] Jamie Callan. *Distributed information retrieval*. In Advances in information retrieval, pages 127–150. Springer, 2000. (Cited on page 29.)

[Cambazoglu 2010] Berkant Barla Cambazoglu, Flavio P Junqueira, Vassilis Plachouras, Scott Banachowski, Baoqiu Cui, Swee Lim and Bill Bridge. *A refreshing perspective of search engine caching*. In Proceedings of the 19th international conference on World wide web, pages 181–190. ACM, 2010. (Cited on pages 2, 50 and 195.)

[CareerJet 2014] CareerJet. *Careerjet.com - Jobs & Careers*, 2014. Offline; accessed 30-September-2014. (Cited on page 178.)

[Ceppi 2012] Sofia Ceppi, Enrico H Gerding and Nicola Gatti. *Merging multiple information sources in federated sponsored search auctions*. In Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3, pages 1323–1324. International Foundation for Autonomous Agents and Multiagent Systems, 2012. (Cited on page 203.)

[Ceri 2010] Stefano Ceri, Daniele Braga, Francesco Corcoglioniti, Michael Grossniklaus and Salvatore Vadacca. Search computing challenges and directions. Springer, 2010. (Cited on pages 1 and 2.)

[Chang 2006] Kevin Chen-Chuan Chang and Junghoo Cho. *Accessing the web: from search to integration.* In Proceedings of the 2006 ACM SIGMOD international conference on Management of data, pages 804–805. ACM, 2006. (Cited on page 30.)

[Chaudhuri 1995] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos and Kyuseok Shim. *Optimizing queries with materialized views.* In 2013 IEEE 29th International Conference on Data Engineering (ICDE), pages 190–190. IEEE Computer Society, 1995. (Cited on page 27.)

[Cohen 2002] William W Cohen and Jacob Richman. *Learning to match and cluster large high-dimensional data sets for data integration.* In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 475–480. ACM, 2002. (Cited on page 49.)

[Cohn 1994] David Cohn, Les Atlas and Richard Ladner. *Improving generalization with active learning.* Machine learning, vol. 15, no. 2, pages 201–221, 1994. (Cited on page 49.)

[Confalonieri 2004] Roberto Confalonieri, John Domingue and Enrico Motta. *Orchestration of semantic web services in IRS-III.* 2004. (Cited on page 21.)

[Craswell 2004] Nick Craswell, Francis Crimmins, David Hawking and Alistair Moffat. *Performance and cost tradeoffs in web search.* In Proceedings of the 15th Australasian database conference-Volume 27, pages 161–169. Australian Computer Society, Inc., 2004. (Cited on page 4.)

[Daniel 2006] Florian Daniel and Barbara Pernici. *Insights into web service orchestration and choreography.* International Journal of E-Business Research, vol. 2, no. 1, pages 58–77, 2006. (Cited on page 21.)

[Das 2006] Gautam Das, Dimitrios Gunopulos, Nick Koudas and Dimitris Tsirogiannis. *Answering top-k queries using views.* In Proceedings

of the 32nd international conference on Very large data bases, pages 451–462. VLDB Endowment, 2006. (Cited on page 48.)

[DBLife 2014] DBLife. *DBLife Frontpage*, 2014. Offline; accessed 30-September-2014. (Cited on page 36.)

[De Backer 2004] Manu De Backer. *On the verification of web services compatibility: A Petri net approach*. In On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops, pages 810–821. Springer, 2004. (Cited on page 23.)

[DeepWeb 2009] DeepWeb. *CompletePlanet - BrightPlanet*, 2009. Offline; accessed 30-September-2009. (Cited on page 28.)

[DeWitt 1990] David J DeWitt, Shahram Ghandeharizadeh, Donovan Schneider, Allan Bricker, Hui-I Hsiao, Rick Rasmussen*et al.* *The Gamma database machine project*. Knowledge and Data Engineering, IEEE Transactions on, vol. 2, no. 1, pages 44–62, 1990. (Cited on page 20.)

[Dey 1998] Debabrata Dey, Sumit Sarkar and Prabuddha De. *Entity matching in heterogeneous databases: a distance-based decision model*. In System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on, volume 7, pages 305–313. IEEE, 1998. (Cited on page 49.)

[Doan 2006] AnHai Doan, Raghu Ramakrishnan and Shivakumar Vaithyanathan. *Managing information extraction: state of the art and research directions*. In Proceedings of the 2006 ACM SIGMOD international conference on Management of data, pages 799–800. ACM, 2006. (Cited on page 30.)

[Du Bois Jr 1969] NS D'Andrea Du Bois Jr. *A solution to the problem of linking multivariate documents*. Journal of the American Statistical Association, vol. 64, no. 325, pages 163–174, 1969. (Cited on page 48.)

[Duda 1973] Peter E Duda and O Richard. *Hart, Pattern Classification and Scene Analysis*, 1973. (Cited on page 48.)

[Duschka 1997a] Oliver M. Duschka and Michael R. Genesereth. *Query planning in infomaster*. In Proceedings of the 1997 ACM symposium on Applied computing, SAC '97, pages 109–111, New York, NY, USA, 1997. ACM. (Cited on pages 3, 17 and 19.)

[Duschka 1997b] Oliver M Duschka and Michael R Genesereth. *Query planning in infomaster*. In Proceedings of the 1997 ACM symposium on Applied computing, pages 109–111. ACM, 1997. (Cited on page 19.)

[Dustdar 2005] Schahram Dustdar and Wolfgang Schreiner. *A survey on web services composition*. International journal of web and grid services, vol. 1, no. 1, pages 1–30, 2005. (Cited on page 20.)

[Easley 2010] David Easley and Jon Kleinberg. Networks, crowds, and markets: Reasoning about a highly connected world. Cambridge University Press, 2010. (Cited on page 187.)

[Edreams 2014] Edreams. *Cheap flights, hotels and vacation packages - eDreams US*, 2014. Online; accessed 30-September-2014. (Cited on page 36.)

[Elfeky 2002] Mohamed G Elfeky, Vassilios S Verykios and Ahmed K Elmagarmid. *TAILOR: A record linkage toolbox*. In Data Engineering, 2002. Proceedings. 18th International Conference on, pages 17–28. IEEE, 2002. (Cited on page 49.)

[EvenDar 2007] Eyal EvenDar, Michael Kearns and Jennifer Wortman. *Sponsored search with contexts*. In Internet and Network Economics, pages 312–317. Springer, 2007. (Cited on page 4.)

[Expedia 2014] Expedia. *Expedia Travel: Vacations, Cheap Flights, Airline Tickets ...*, 2014. Online; accessed 30-September-2014. (Cited on page 36.)

[Fagin 2003] Ronald Fagin, Amnon Lotem and Moni Naor. *Optimal aggregation algorithms for middleware*. Journal of Computer and System Sciences, vol. 66, no. 4, pages 614–656, 2003. (Cited on page 48.)

[Fagni 2006] Tiziano Fagni, Raffaele Perego, Fabrizio Silvestri and Salvatore Orlando. *Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data.* ACM Transactions on Information Systems (TOIS), vol. 24, no. 1, pages 51–78, 2006. (Cited on page 26.)

[Feder 1995] Tomás Feder and Rajeev Motwani. *Clique partitions, graph compression and speeding-up algorithms.* Journal of Computer and System Sciences, vol. 51, no. 2, pages 261–272, 1995. (Cited on page 28.)

[Friedman 1997] Marc Friedman and Daniel S Weld. *Efficiently executing information-gathering plans.* In In Proc. of the Int. Joint Conf. of AI (IJCAI. Citeseer, 1997. (Cited on page 19.)

[Garcia 2007] Steven Garcia. *Search engine optimisation using past queries.* PhD thesis, PhD thesis, RMIT University, 2007. (Cited on page 26.)

[Garofalakis 2001] Minos N Garofalakis and Phillip B Gibbons. *Approximate Query Processing: Taming the TeraBytes.* In VLDB, 2001. (Cited on page 27.)

[GigaAlert 2014] GigaAlert. *Giga Alert - Professional Web Alerts*, 2014. Online; accessed 30-September-2014. (Cited on page 29.)

[Gilbert 2004] Anna C Gilbert and Kirill Levchenko. *Compressing network graphs.* In Proceedings of the LinkKDD workshop at the 10th ACM Conference on KDD, 2004. (Cited on page 28.)

[Guha 1998] Ramanathan V Guha, Ora Lassila, Eric Miller and Dan Brickley. *Enabling inferencing.* 1998. (Cited on page 19.)

[Guha 2004] Sudipto Guha, Nick Koudas, Amit Marathe and Divesh Srivastava. *Merging the results of approximate match operations.* In Proceedings of the Thirtieth international conference on Very large data bases-Volume 30, pages 636–647. VLDB Endowment, 2004. (Cited on page 49.)

[Guntzer 2001] J Guntzer, W-T Balke and Werner Kießling. *Towards efficient multi-feature queries in heterogeneous environments.* In Information Technology: Coding and Computing, 2001. Proceedings. International Conference on, pages 622–628. IEEE, 2001. (Cited on page 48.)

[Gupta 1993] Ashish Gupta, Inderpal Singh Mumick and V. S. Subrahmanian. *Maintaining views incrementally.* SIGMOD Rec., vol. 22, pages 157–166, June 1993. (Cited on page 50.)

[Gupta 1995] Ashish Gupta and Inderpal Singh Mumick. *Maintenance of Materialized Views: Problems, Techniques, and Applications.* 1995. (Cited on pages 50 and 194.)

[Gupta 1999] Ashish Gupta and Iderpal Singh Mumick. Materialized views: techniques, implementations, and applications. MIT press, 1999. (Cited on pages 6 and 16.)

[Haas 2004] Peter J Haas and Christian Koenig. *A bi-level Bernoulli scheme for database sampling.* In Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pages 275–286. ACM, 2004. (Cited on page 27.)

[Halevy 2001] Alon Y. Halevy. *Answering queries using views: A survey.* The VLDB Journal, vol. 10, pages 270–294, December 2001. (Cited on page 44.)

[Hamadi 2003] Rachid Hamadi and Boualem Benatallah. *A Petri net-based model for web service composition.* In Proceedings of the 14th Australasian database conference-Volume 17, pages 191–200. Australian Computer Society, Inc., 2003. (Cited on pages 23, 24 and 25.)

[Hastings 1970] W Keith Hastings. *Monte Carlo sampling methods using Markov chains and their applications.* Biometrika, vol. 57, no. 1, pages 97–109, 1970. (Cited on page 27.)

[Hristidis 2004] Vagelis Hristidis and Yannis Papakonstantinou. *Algorithms and applications for answering ranked queries using ranked views.* The VLDB Journal, vol. 13, pages 49–70, January 2004. (Cited on page 48.)

[Hull 1993] David Hull. *Using statistical testing in the evaluation of retrieval experiments.* In Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '93, pages 329–338, New York, NY, USA, 1993. ACM. (Cited on page 51.)

[Ilyas 2004] Ihab F Ilyas, Walid G Aref and Ahmed K Elmagarmid. *Supporting top-k join queries in relational databases.* The VLDB Journalï¿½The International Journal on Very Large Data Bases, vol. 13, no. 3, pages 207–221, 2004. (Cited on pages 36 and 63.)

[IndexAdvisor 2014] IndexAdvisor. *IndexAdvisor*, 2014. Online; accessed August-2015. (Cited on page 6.)

[InternetArchive 2014] InternetArchive. *Internet Archive: Digital Library of Free Books, Movies ...*, 2014. Online; accessed 30-September-2014. (Cited on page 28.)

[Ipeirotis 2001] Panagiotis G Ipeirotis, Luis Gravano and Mehran Sahami. *Probe, count, and classify: categorizing hidden web databases.* In ACM SIGMOD Record, volume 30, pages 67–78. ACM, 2001. (Cited on page 28.)

[Ives 1999] Zachary G Ives, Daniela Florescu, Marc Friedman, Alon Levy and Daniel S Weld. *An adaptive query execution system for data integration.* In ACM SIGMOD Record, volume 28, pages 299–310. ACM, 1999. (Cited on page 17.)

[Ives 2004] Zachary G Ives, Alon Y Halevy and Daniel S Weld. *Adapting to source properties in processing data integration queries.* In Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pages 395–406. ACM, 2004. (Cited on page 20.)

[Jin 2011] Xin Jin, Nan Zhang and Gautam Das. *Attribute domain discovery for hidden web databases.* In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, pages 553–564. ACM, 2011. (Cited on pages 30 and 112.)

[Jonsson 1996] Bjorn Jonsson, Divesh Srivastava, S Dar*et al. Semantic data caching and replacement.* In Proc of the 22nd Intl Conf on VLDB. Bombay, India, 1996. (Cited on page 3.)

[Jónsson 1998] Björn T Jónsson, Michael J Franklin and Divesh Srivastava. *Interaction of query evaluation and buffer management for information retrieval.* In ACM SIGMOD Record, volume 27, pages 118–129. ACM, 1998. (Cited on page 26.)

[Keller 1996] Arthur M Keller and Julie Basu. *A predicate-based caching scheme for client-server database architectures.* The VLDB Journalï¿½The International Journal on Very Large Data Bases, vol. 5, no. 1, pages 035–047, 1996. (Cited on page 3.)

[Kónig 2008] Dieter Kónig, Niels Lohmann, Simon Moser, Christian Stahl and Karsten Wolf. *Extending the compatibility notion for abstract WS-BPEL processes.* In Proceedings of the 17th international conference on World Wide Web, pages 785–794. ACM, 2008. (Cited on page 25.)

[Krishnamurthy 2005] Vaishnavi Krishnamurthy, Michalis Faloutsos, Marek Chrobak, Li Lao, J-H Cui and Allon G Percus. *Reducing large internet topologies for faster simulations.* In NETWORKING 2005. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems, pages 328–341. Springer, 2005. (Cited on page 28.)

[Kwok 1996] Chung T Kwok, Daniel S Weld*et al. Planning to gather information.* In PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, pages 32–39. Citeseer, 1996. (Cited on pages 3, 17 and 18.)

[Lambrecht 1999] Eric Lambrecht, Subbarao Kambhampati and Senthil Gnanaprakasam. *Optimizing recursive information gathering plans.* In IJCAI, volume 99, pages 1204–1211, 1999. (Cited on pages 3, 17 and 19.)

[Lawrence 1998] Steve Lawrence and C Lee Giles. *Searching the world wide web.* Science, vol. 280, no. 5360, pages 98–100, 1998. (Cited on page 28.)

[Lawrence 1999] Steve Lawrence and C Lee Giles. *Accessibility of information on the web.* Nature, vol. 400, no. 6740, pages 107–107, 1999. (Cited on page 28.)

[Lempel 2003] Ronny Lempel and Shlomo Moran. *Predictive caching and prefetching of query results in search engines.* In Proceedings of the 12th international conference on World Wide Web, pages 19–28. ACM, 2003. (Cited on page 26.)

[Leskovec 2006] Jure Leskovec and Christos Faloutsos. *Sampling from large graphs.* In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 631–636. ACM, 2006. (Cited on pages 28 and 186.)

[Levy 1996] Alon Levy, Anand Rajaraman and Joann Ordille. *Querying heterogeneous information sources using source descriptions.* 1996. (Cited on pages 3, 17 and 18.)

[Madhavan 2008] Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen and Alon Halevy. *Google's Deep Web crawl.* Proc. VLDB Endow., vol. 1, no. 2, pages 1241–1252, 2008. (Cited on pages 1, 3, 7, 30, 32, 41, 44, 45, 46 and 83.)

[Manolescu 2001] Ioana Manolescu, Daniela Florescu and Donald Kossmann. *Answering XML Queries on Heterogeneous Data Sources.* In VLDB, volume 1, pages 241–250, 2001. (Cited on page 17.)

[Markatos 2001] Evangelos P. Markatos. *On caching search engine query results.* Computer Communications, vol. 24, no. 2, pages 137–143, 2001. (Cited on page 26.)

[Martens 2005a] Axel Martens. *Analyzing web service based business processes.* In Fundamental Approaches to Software Engineering, pages 19–33. Springer, 2005. (Cited on page 24.)

[Martens 2005b] Axel Martens. *Consistency between executable and abstract processes.* In e-Technology, e-Commerce and e-Service, 2005. EEE'05. Proceedings. The 2005 IEEE International Conference on, pages 60–67. IEEE, 2005. (Cited on pages 24 and 25.)

[Metropolis 1953] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller and Edward Teller. *Equation of state calculations by fast computing machines.* The journal of chemical physics, vol. 21, no. 6, pages 1087–1092, 1953. (Cited on page 27.)

[Milanovic 2004] Nikola Milanovic and Miroslaw Malek. *Current solutions for web service composition.* IEEE Internet Computing, no. 6, pages 51–59, 2004. (Cited on page 20.)

[Monge 1996] Alvaro E Monge, Charles Elkan*et al. The Field Matching Problem: Algorithms and Applications.* In KDD, pages 267–270, 1996. (Cited on page 49.)

[Murata 1989] Tadao Murata. *Petri nets: Properties, analysis and applications.* Proceedings of the IEEE, vol. 77, no. 4, pages 541–580, 1989. (Cited on pages 10, 116, 135, 198 and 199.)

[Narayanan 2002] Srini Narayanan and Sheila A McIlraith. *Simulation, verification and automated composition of web services.* In Proceedings of the 11th international conference on World Wide Web, pages 77–88. ACM, 2002. (Cited on page 23.)

[Naughton 2001] Jeffrey F. Naughton, David J. DeWitt, David Maier, Ashraf Aboulnaga, Jianjun Chen, Leonidas Galanis, Jaewoo Kang, Rajasekar

Krishnamurthy, Qiong Luo, Naveen Prakash*et al.* *The Niagara internet query system*. IEEE Data Eng. Bull., vol. 24, no. 2, pages 27–33, 2001. (Cited on page 17.)

[Nepal 1999] Surya Nepal and MV Ramakrishna. *Query processing issues in image (multimedia) databases*. In Data Engineering, 1999. Proceedings., 15th International Conference on, pages 22–29. IEEE, 1999. (Cited on page 48.)

[Nezhad 2007] Motahari Nezhad, Hamid Reza, Boualem Benatallah, Axel Martens, Francisco Curbera and Fabio Casati. *Semi-automated adaptation of service interactions*. In Proceedings of the 16th international conference on World Wide Web, pages 993–1002. ACM, 2007. (Cited on page 24.)

[OASIS 2007] OASIS. *OASIS: Web Services Business Process Execution Language. Technical report*. Rapport technique, 2007. (Cited on page 21.)

[Olken 1993] Frank Olken. *Random sampling from databases*. PhD thesis, University of California, 1993. (Cited on page 27.)

[Ouyang 2005] Chun Ouyang, Eric Verbeek, Wil MP van der Aalst, Stephan Breutel, Marlon Dumas and Arthur HM ter Hofstede. *WofBPEL: A tool for automated analysis of BPEL processes*. In Service-Oriented Computing-ICSOC 2005, pages 484–489. Springer, 2005. (Cited on page 24.)

[Ouyang 2007] Chun Ouyang, Eric Verbeek, Wil MP Van Der Aalst, Stephan Breutel, Marlon Dumas and Arthur HM Ter Hofstede. *Formal semantics and analysis of control flow in WS-BPEL*. Science of Computer Programming, vol. 67, no. 2, pages 162–198, 2007. (Cited on page 24.)

[Özsu 2011] M Tamer Özsu and Patrick Valduriez. Principles of distributed database systems. Springer Science & Business Media, 2011. (Cited on page 20.)

[Pasula 2002] Hanna Pasula, Bhaskara Marthi, Brian Milch, Stuart Russell and Ilya Shpitser. *Identity uncertainty and citation matching.* In Advances in neural information processing systems, pages 1401–1408, 2002. (Cited on page 49.)

[Peralta 2006] Veronica Peralta. *Data freshness and data accuracy: A state of the art.* Rapport technique, Universidad de la Republica, Uruguay, 2006. (Cited on page 194.)

[Peralta 2008] Verónika Peralta. *Data quality evaluation in data integration systems.* PhD thesis, Citeseer, 2008. (Cited on page 7.)

[Peterson 1981] James L Peterson. *Petri net theory and the modeling of systems.* 1981. (Cited on page 22.)

[Petri 1962] Carl Adam Petri. *Kommunikation mit automaten.* 1962. (Cited on page 22.)

[Piatetsky-Shapiro 1984] Gregory Piatetsky-Shapiro and Charles Connell. *Accurate estimation of the number of tuples satisfying a condition.* In ACM SIGMOD Record, volume 14, pages 256–276. ACM, 1984. (Cited on page 27.)

[Polyzotis 2011] Neoklis Polyzotis. The rank join problem. Springer, 2011. (Cited on pages 36 and 63.)

[Prudhommeaux 2005] Eric Prudhommeaux and Andy Seaborne. *SPARQL Query Language for RDF http://www. w3. org.* Rapport technique, TR/rdf-sparql-query, 2005. (Cited on page 20.)

[Rafiei 2005] Davood Rafiei. *Effectively visualizing large networks through sampling.* In Visualization, 2005. VIS 05. IEEE, pages 375–382. IEEE, 2005. (Cited on page 28.)

[Raghavan 2000] Sriram Raghavan and Hector Garcia-Molina. *Crawling the hidden web.* 2000. (Cited on pages 28, 29, 30 and 84.)

[Rajaraman 1995] Anand Rajaraman, Yehoshua Sagiv and Jeffrey D Ullman. *Answering queries using templates with binding patterns*. In Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pages 105–112. ACM, 1995. (Cited on page 18.)

[Rajaraman 2009] Anand Rajaraman. *Kosmix: high-performance topic exploration using the deep web*. Proceedings of the VLDB Endowment, vol. 2, no. 2, pages 1524–1529, 2009. (Cited on page 37.)

[Ravikumar 2004] Pradeep Ravikumar and William W Cohen. *A hierarchical graphical model for record linkage*. In Proceedings of the 20th conference on Uncertainty in artificial intelligence, pages 454–461. AUAI Press, 2004. (Cited on page 49.)

[Robie 2001] Jonathan Robie and Ratrick Lehti. *Updates in XQuery*. In XML Conference & Exhibiton, 2001. (Cited on page 20.)

[Saraiva 2001] Paricia Correia Saraiva, Edleno Silva de Moura, Novio Ziviani, Wagner Meira, Rodrigo Fonseca and Berthier Riberio-Neto. *Rank-preserving two-level caching for scalable search engines*. In Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, pages 51–58. ACM, 2001. (Cited on page 26.)

[Schafer 1999] J Ben Schafer, Joseph Konstan and John Riedl. *Recommender systems in e-commerce*. In Proceedings of the 1st ACM conference on Electronic commerce, pages 158–166. ACM, 1999. (Cited on page 202.)

[Shokouhi 2006] Milad Shokouhi, Justin Zobel, Falk Scholer and Seyed MM Tahaghoghi. *Capturing collection size for distributed non-cooperative retrieval*. In Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, pages 316–323. ACM, 2006. (Cited on page 30.)

[SOCareers 2014] SOCareers. *Stack Overflow Careers 2.0*, 2014. Online; accessed 30-September-2014. (Cited on page 178.)

[Srivastava 2006] Utkarsh Srivastava, Kamesh Munagala, Jennifer Widom and Rajeev Motwani. *Query optimization over web services.* In Proceedings of the 32nd international conference on Very large data bases, pages 355–366. VLDB Endowment, 2006. (Cited on page 21.)

[Suchanek 2007] Fabian M Suchanek, Gjergji Kasneci and Gerhard Weikum. *Yago: a core of semantic knowledge.* In Proceedings of the 16th international conference on World Wide Web, pages 697–706. ACM, 2007. (Cited on page 56.)

[Tan 2009] Wei Tan, Yushun Fan and MengChu Zhou. *A petri net-based method for compatibility analysis and composition of web services in business process execution language.* Automation Science and Engineering, IEEE Transactions on, vol. 6, no. 1, pages 94–106, 2009. (Cited on page 24.)

[Tatemura 2007] Junichi Tatemura, Arsany Sawires, Oliver Po, Songting Chen, K Selcuk Candan, Diviyakant Agrawal and Maria Goveas. *Mashup feeds: continuous queries over web services.* In Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pages 1128–1130. ACM, 2007. (Cited on page 21.)

[Tejada 2002] Sheila Tejada, Craig A Knoblock and Steven Minton. *Learning domain-independent string transformation weights for high accuracy object identification.* In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 350–359. ACM, 2002. (Cited on page 49.)

[TripAdvisor 2014] TripAdvisor. *Travelers' Choice - TripAdvisor - Best Beaches, Family ...*, 2014. Online; accessed 30-September-2014. (Cited on page 36.)

[Trombetta 2004] Alberto Trombetta and Danilo Montesi. *Equivalences and optimizations in an expressive XSLT fragment.* In Database Engineering and Applications Symposium, 2004. IDEAS'04. Proceedings. International, pages 171–180. IEEE, 2004. (Cited on page 20.)

[Tsatalos 1996] Odysseas G Tsatalos, Marvin H Solomon and Yannis E Ioannidis. *The GMAP: A versatile tool for physical data independence*. The VLDB Journal, vol. 5, no. 2, pages 101–118, 1996. (Cited on page 16.)

[Twitter 2014] Twitter. *Twitter Developers*, 2014. Online; accessed 30-September-2014. (Cited on page 107.)

[Valduriez 1987] Patrick Valduriez. *Join indices*. ACM Transactions on Database Systems (TODS), vol. 12, no. 2, pages 218–246, 1987. (Cited on page 16.)

[Van der Aalst 1998] Wil MP Van der Aalst. *The application of Petri nets to workflow management*. Journal of circuits, systems, and computers, vol. 8, no. 01, pages 21–66, 1998. (Cited on page 24.)

[Verykios 2000] Vassilios S Verykios, Ahmed K Elmagarmid and Elias N Houstis. *Automating the approximate record-matching process*. Information sciences, vol. 126, no. 1, pages 83–98, 2000. (Cited on page 49.)

[Verykios 2004] Vassilios S Verykios and George V Moustakides. *A generalized cost optimal decision model for record matching*. In Proceedings of the 2004 international workshop on Information quality in information systems, pages 20–26. ACM, 2004. (Cited on page 48.)

[Vitter 1985] Jeffrey S Vitter. *Random sampling with a reservoir*. ACM Transactions on Mathematical Software (TOMS), vol. 11, no. 1, pages 37–57, 1985. (Cited on page 27.)

[Von Neumann 1951] John Von Neumann. *Various techniques used in connection with random digits*. Applied Math Series, vol. 12, no. 36-38, page 1, 1951. (Cited on page 27.)

[Voorhees 1999] Ellen M Voorhees and Donna K Hardman. *The {Eight Text Retrieval Conference}{(TREC-8)}*. 1999. (Cited on page 51.)

[Voorhees 2005] Ellen M Voorhees. *The TREC robust retrieval track*. In ACM SIGIR Forum, volume 39, pages 11–20. ACM, 2005. (Cited on page 51.)

[Winkler 2002] William E Winkler. *Methods for record linkage and bayesian networks*. Rapport technique, Technical report, Statistical Research Division, US Census Bureau, Washington, DC, 2002. (Cited on page 48.)

[WSCDL 2005] WSCDL. *W3C: Web Services Choreography Description Language, Version 1.0. W3C Candidate Recommendation*, 2005. (Cited on page 21.)

[WSCI 2002] WSCI. *W3C: Web Service Choreography Interface (WSCI), Version 1.0. W3C Note.*, 2002. (Cited on page 21.)

[WSMO 2010] WSMO. *WSMO: Web Service Modeling Ontology*, 2010. (Cited on page 21.)

[Wu 2006] Ping Wu, Ji-Rong Wen, Huan Liu and Wei-Ying Ma. *Query Selection Techniques for Efficient Crawling of Structured Web Sources*. Data Engineering, International Conference on, vol. 0, page 47, 2006. (Cited on pages 7, 31, 32, 44, 46, 70, 73, 90, 112 and 175.)

[Xie 2002] Yinglian Xie and David O'Hallaron. *Locality in search engine queries and its implications for caching*. In INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, volume 3, pages 1238–1247. IEEE, 2002. (Cited on page 26.)

[YahooPipes 2014] YahooPipes. *Yahoo! Pipes*, 2014. Online; accessed August-2015. (Cited on page 21.)

[YahooTravel 2014] YahooTravel. *Yahoo! Travel*, 2014. Online; accessed 30-September-2014. (Cited on page 36.)

[Yancey 2005] William E Yancey. *Evaluating string comparator performance for record linkage*. Statistical Research Division Research Report, http://www. census. gov/srd/papers/pdf/rrs2005-05. pdf, 2005. (Cited on page 49.)

[Yang 1987] HZ Yang and Per-Åke Larson. *Query Transformation for PSJ-Queries.* In vldb, pages 245–254, 1987. (Cited on page 16.)

[YelpNZ 2014] YelpNZ. *Yelp New Zealand, Reviews*, 2014. Online; accessed 30-September-2014. (Cited on pages 42 and 46.)

[Yi 2003] Ke Yi, Hai Yu, Jun Yang, Gangqiang Xia and Yuguo Chen. *Efficient Maintenance of Materialized Top-k Views.* Data Engineering, International Conference on, vol. 0, page 189, 2003. (Cited on page 50.)

[Yu 2004] Cong Yu and Lucian Popa. *Constraint-based XML query rewriting for data integration.* In Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pages 371–382. ACM, 2004. (Cited on page 17.)

[Zagorac 2014] Srđan Zagorac and Russel Pears. *Web Materialization Formulation: Modelling Feasible Solutions.* In Database and Expert Systems Applications, pages 366–374. Springer, 2014. (Cited on page 9.)

[Zerfos 2005] Petros Zerfos, Junghoo Cho and Alexandros Ntoulas. *Downloading textual hidden web content through keyword queries.* Digital Libraries, Joint Conference on, vol. 0, pages 100–109, 2005. (Cited on pages 7, 31, 32, 44 and 46.)

[Zhang 2008] Jiangong Zhang, Xiaohui Long and Torsten Suel. *Performance of compressed inverted list caching in search engines.* In Proceedings of the 17th international conference on World Wide Web, pages 387–396. ACM, 2008. (Cited on page 26.)

[Zhou 2007] Jingren Zhou, Per-Ake Larson and Hicham G. Elmongui. *Lazy maintenance of materialized views.* In Proceedings of the 33rd international conference on Very large data bases, VLDB '07, pages 231–242, 2007. (Cited on page 50.)

[Zhuge 1995] Yue Zhuge, Héctor García-Molina, Joachim Hammer and Jennifer Widom. *View maintenance in a warehousing environment.* SIGMOD Rec., vol. 24, pages 316–327, May 1995. (Cited on page 50.)