PAY-AS-YOU-GO INSTANCE-LEVEL INTEGRATION

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2016

By
Ruhaila Maskat
Computer Science

Contents

List of Tables

List of Figures

Abstract

With the growing demand for information in various domains, sharing of information from heterogeneous data sources is now a necessity. Data integration approaches promise to combine data from these different sources and present to the user a single, unified view of these data. However, although these approaches offer high quality services for the managing and integrating of data, they come with a high cost. This is because a great amount of manual effort to form relationships across data sources is needed to set up the data integration system. A newer variant of data integration, known as *dataspaces*, aims to spread the large manual effort spent at the start of the data integration system to the rest of the system's phases. This is achieved by soliciting from the user their feedback on a chosen artefact of a dataspace, either by explicit ways or implicitly. This practice is known as *pay-as-you-go*, where a user continuously *pays* to the data integration system, by providing feedback, to gain improvements in the quality of data integration.

This PhD addresses two challenges in data integration by using pay-as-you-go approaches. The first is to identify instances relevant to a user's information need, calling for semantic mappings to be closely considered. Our contribution is a technique that ranks mappings with the help of implicit user feedback (i.e., terms found in query logs). Our evaluation shows that to produce stable rankings, our technique does not require large-sized query logs, and that our generated ranking is able to respond satisfactorily to the amount of terms inclined towards a particular data source, where we describe it as *skew*. The second challenge that we address is the identification of duplicate instances from disparate data sources. We contribute a strategy that uses explicitly-obtained user feedback to drive an evolutionary search algorithm to find suitable parameters for an underlying clustering algorithm. Our experiments show that optimising the algorithm's parameters and introducing attribute weights produces fitter clusters than clustering alone. However, our strategy to improve on integration quality can be quite expensive. Therefore, we propose a pruning technique to select from a dataset any records that are informative. Our experiment shows that on most of the datasets, our pruner produce comparably fit clusters with more feedback received.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

Deepest thanks to my parents, the rock and lighthouse in my life.

I would also like to thank my supervisors Prof. Norman W. Paton and Dr Suzanne M. Embury for their guidance throughout this long journey. I have learnt much from both.

Last but not least, thank you to the friends and people whom I have met and gotten to know during my stay in Manchester.

Chapter 1

Introduction

Sharing of information can improve understanding. Relying on a single data source to make decisions can no longer fulfil today's demands for information in many commerce, scientific and leisure applications. Hence, it is necessary to use data from a collection of data sources. Data sources include all forms of repositories, for example, be it computer files, data streams or databases. For example, it is important that a potential creditor accesses and collects information of a loan applicant from several independent data sources, that are maintained by other creditors and law enforcers, before deciding to extend credit.

This proposal of engaging with multiple data sources is widely known as *data integration*. The primary goal of data integration is to collect and display data from multiple sources making them appear to come from a single source [Hal01, Len02]. The benefit of this is that users are relieved from having to know *how* to retrieve the information they seek, and can focus on *what* information they require. This research focuses on sources that are in the form of relational data structures, but the techniques investigated can be adapted for use with other types of source.

1.1 PhD Motivation

There are numerous challenges in data integration. This PhD tackles two of them. The first challenge is to identify multiple representations or instances of single real-world objects across data sources. This is known variously as *instance integration*, *duplicate detection*, *entity resolution* or *record linkage*. In business, heterogeneity can exist when independent organisations are merged. From the example given in Figure 1.1, an instance is considered successfully integrated

when records *CS1* and *45* are identified as similar. Two records are said to be *similar* when they represent a common object in reality.

Often, both merging organisations store information about the same customer; however, the records are likely to have been structured differently. For example, the *Name* of a customer can be stored as a single attribute in one organisation (for example, *Full Name*) but separated into two attributes by the other, (*First Name* and *Last Name*). It is also likely that some information applies to only one of the organisations, for example, the information about *Country*. Prior to merging, an organisation's scope of business may be local; thus, a *Country* attribute was irrelevant. These are examples of *schematic heterogeneity* [KS91].

When viewed at the data level, heterogeneity occurs when there are differences in the values of similar records contained in the same or different data sources. These differences can pose a challenge when trying to identify similar instances. *Data heterogeneity* found in our example are *Hartley Road* and *Hartley Rd.*, which exhibits the use of abbreviations. Synonymous words (for example, *dry* and *desiccated*) also add to the heterogeneity of data. Another example is the month *June* which can be represented with the number *6*. It is quite common that names can be written down in more than one convention, for example, the names *John Smith* and *John B. Smith* can refer to the same person.

Data heterogeneity is the problem that is the focus of this research; however, other variants of *data heterogeneity*, for example, *mirror detection* and *anaphora resolution*, are not. *Mirror detection* [CSGM00, BGMZ97] involves the identification of similar or identical web pages, while *anaphora resolution* [Mit14] tries to resolve references of a noun or pronoun. For example, the noun *President of America* refers to *Barack Obama*.

The second challenge is in the identification of instances that can be relevant to a user's information need. Consider a manager in a newly-merged organisation who intends to become familiar with their recently-expanded customer base. She posts a query, expressing what information she wishes to investigate. Her

Organisation 1: Customer

ID	FirstName	LastName	DateOfBirth	Block	Level	Unit	Street	Postcode
CS1	John	Smith	3/6/75	17	2	6	Hartley Road	M168PA
CS2	John	Smith	6/3/75	18	2	6	Hartley Road	M168PA

Organisation 2: Buyer

ID	Name	DOB	Lot	Street	Postcode	Country
45	John B. Smith	3 June 75	17-02-06	Hartley Rd.	M16 8PA	United Kingdom

Figure 1.1: Schemas that show schematic and data heterogeneity.

expectation would be to receive a set of instances/records that agrees with what she knows about the entities in her domain of choice. *Precise* knowledge of what information she expects can be guesswork, as only she knows exactly what she requires. Therefore, a data integration system could only attempt to present to her *"relevant"* results based on her posted query. A relevant result should contain records that are closely connected to her information need.

Underlying a data integration system, hidden from the user, is a schema that presents a reconciled view of data from multiple sources [Len02]. This schema is generally known as a *global schema* (Figure 1.2). In order for data to be retrieved, the semantic relationship between the attributes in the source schemas and the global schema must first be established. These relationships are captured using semantic mappings (Figure 1.3). Mappings are described using a database *view*, which is effectively a virtual relation. In data integration, the number of mappings produced can be large [Hal01], suggesting a large amount of records returned with the result set. This not only widens the search space for finding relevant records, but also for integrating instances. Identifying instances relevant to a user's information needs in data integration calls for semantic mappings to be taken into consideration.

1.2 Pay-As-You-Go Data Integration Approach (Dataspaces)

This section introduces the key features of pay-as-you-go data integration as a newer variant of classical data integration. We start by defining the different

Global schema: Cust

ID	FirstName	LastName	Block	Level	Unit	Street	Postcode	Country
CS1	John	Smith	17	2	6	Hartley Road	M168PA	UK
CS2	John	Smith	18	2	6	Hartley Road	M168PA	UK

Figure 1.2: A global schema.

```
Mapping 1:
select firstname, lastname, block, level, unit, street, postcode
from customer

Mapping 2:
select name, lot, street, postcode, country
from buyer
```

Figure 1.3: Semantic mappings between schemas in Figures 1.1 and 1.2.

types of source that constitute a classical data integration, provide a description of its life cycle, and state the problems that motivated our proposal for pay-as-you-go data integration. Then, we describe pay-as-you-go data integration, its inherited and distinguishing principles, its life cycle and challenges. We end this section with a discussion of the gaps in pay-as-you-go data integration that our research fills.

To manually collect and integrate data from disparate sources is a laborious task. An extensive knowledge of each underlying data source is needed to successfully achieve this, in both the domain area (i.e., the semantics of the schemas) and technically (i.e., formal language expression). However, this usually results in a great expenditure of effort on *how* to integrate data, instead of on *what* data is required to accomplish the task at hand. Having a single view of a collection of data sources would free users from the tedious job of accessing each source, retrieving records using its underlying query language, and manually consolidating the results. Such a single view of data retrieval is the primary aim of data integration.

Behind this single view of data integration lies multiple data sources. They are known as *local sources*, where the actual data records are stored. Schemas in these sources are known as *local schemas*. A unified view of these *local sources* is presented by a *global schema*.

Data integration involves several phases [Lev98, DU00]:

i *Register data sources. Register* refers to the granting of any authorisations to access the data and structure of local sources. In this phase, middleware (for example, wrapper and mediator) that produces a seamless link between individual local sources and a data integration management system is constructed [Lev98, DU00].

ii *Identify schema matchings.* A schema matching represents a semantic relationship between elements across local schemas [RB01]. This includes identifying similar concepts in the real world, for example, *buyer* and *customer*, and identifying their corresponding attributes for example, *gender* and *sex*, *county* and *state*.

iii *Form semantic mappings.* While matches deal with local schemas, mappings describe the semantic association between a set of local schemas and a global schema. There are two well-known approaches to mapping formation: global-as-view (GAV) and local-as-view (LAV) [Len02]. With GAV, each element

of the global schema is expressed as a view over a group of local sources. In contrast, LAV begins with the derivation of a global schema, followed by the modelling of local sources as views over the global schema.

iv *Reformulate user query.* Fundamentally, this process involves transforming a user query posed in terms of a global schema into a set of queries executable over local schemas [DU00]. These queries have to reflect the semantic mappings obtained in the previous phase.

v *Resolve instances/entities* that belong in the real world. Access to disparate data sources can give rise to a host of records that represent one common instance (i.e., *duplicates*). However, it is no easy task to determine which instance in the real world that a group of records actually refers to due to the presence of semantic heterogeneity. The result of instance resolution is a set of pairs and groups of records that are perceived by users as being semantically *similar*.

vi *Fuse semantically similar records. Entity resolution* can leave an integration effort with inconsistent similar records. Bleiholder [BN09] listed two causes of this condition of inconsistency. They are attribute values that have been left blank or null, and contradicting non-null values of a semantically similar record pair. Once reconciled, similar records resurface as *"a single, consistent and clean representation"* [BN09] of a real world instance.

A significant drawback in classical data integration is in its high cost to initialise. During initialisation, identifying schema matchings and forming semantic mappings is costly because a great deal of human effort is needed. Such manual exercise can determine exact semantic relationships, and hence, produce highly accurate results; however, it is at the expense of delayed start-up time. This process of defining the semantic relationships of schemas has been termed *semantic integration* [FHM05].

1.2.1 Definition

In order to overcome the drawback of classical data integration, a newer variant of data integration has been proposed. This variant is known as *pay-as-you-go data integration* or *dataspaces* [HFM06]. In a dataspace, full semantic integration at the start of a dataspace's life is not mandatory. A dataspace can accept partial semantic integration, where the integration is performed to a degree that

complies with any known semantic relationships. With recurring conditions, such as addition of new local sources and obtaining of new data in registered local sources, a dataspace lets integration to happen in increments and can be improved whenever there is a need. As a result, the integration cost that previously was borne upfront by a classical data integration, can now be spread throughout the life time of a dataspace, and it is able to start operating earlier than a classical data integration.

Another characteristic of a dataspace is in the query results that it produces and their interplay with users. The results are aimed at being *best-effort* based on current, available local sources. A dataspace uses a user's domain knowledge in order to produce better quality results. This knowledge comes in the form of feedback pinned on usually two artefacts of a dataspace, which includes either *matches* [JFH07, JFH08] or *records of a result set* [HBF+09, HBM+12, BPF+11]. In essence, feedback carries a positive or negative validation of what is suggested by a dataspace, based on the user's perception of truth. When more feedback is supplied, the quality of the returned results improves, acting as an incentive to users to continuously provide feedback.

In the next subsection, we describe the life cycle of a dataspace, where some phases inherit from a classical data integration, and others that are specific to a dataspace.

1.2.2 Life Cycle

Hedeler *et al.* [HBF+09] proposes a life cycle for dataspaces which partly extends classical data integration and in other parts is specific to dataspaces. We alter Hedeler's proposal by separating two essential processes to form two independent phases; namely, instance integration and data fusion, and describe the characteristics of each phase of the life cycle.

i *Identify data sources*, that have been built in various formats (i.e., *structured*, *unstructured* and *semi-structured*) where each is intended for specific interfaces; and is located at different locations (i.e., *local* and *distributed*).

ii *Design/derive a global schema* that reflects data in the underlying set of identified sources. There are three approaches to achieving this. At one end of the spectrum is where users with domain knowledge manually design a global schema for their needs. At the other end, an algorithm searches the sources

to choose and suggest schema elements that may be relevant completely automatically. In the middle, users select from existing schemas and put together a set to form a global schema. Designing and deriving a global schema can seek help from existing artefacts of a dataspace. They include schemas and records of the already integrated local sources, schema matchings and semantic mappings. Unlike manual and semi-automatic means of global schema-forming which can rely on users to make decisions, the fully automated approach can be a challenge, since there are no guarantees that the produced global schema accurately describes the domain of interest. Global schemas can be of two types. When a global schema consists of all the elements of the underlying local schemas but is void of any information about the semantic relationships between them, it is a union-type schema [HBF+09]. In contrast, when there have been processes that combine the elements from different schemas in ways that reflect their semantics, the schema is a merged-schema [HBF+09].

iii *Identifying schema matchings* in dataspaces builds, in some parts, upon classical data integration's matching identification process. Similarly, matchings are formed from semantic relationships across elements of local schemas. Conversely, in dataspaces, either *automatic*, *semi-automatic* or *manual* types of approach can be used to identify matchings. Such automatic techniques give rise to *uncertainty* about the validity of the proposed semantic relationships, where this uncertainty is expressed in the form of *scores*. From Hedeler's survey [HBF+09], information that has so far been used to identify matchings, irrespective of the type of approach used, are schemas, records and training data (if machine learning algorithms are involved).

iv To *derive semantic mappings*, useful components of a dataspace (i.e., local schemas and their records), a dataspace's intermediate product such as earlier identified matchings, and external resources, for example, training data, may be used to produce mappings either by fully human effort, the use of a suitable algorithm or by an algorithm which permits human intervention. Like matchings, the semantic associations between elements of local schemas and the global schema can introduce uncertainty; also, like matchings, these semantic associations can be represented by scores. By now, a dataspace has recognised a set of useful data sources on which user query could be posed.

v *Searching or querying* in dataspace can occur before or after duplicates have been eliminated by recognising similar instances and resolving any conflicts.

Search-related operations that take place during this phase are exploratory in nature, using keywords; while querying-operations are mostly related to the selection, projection, joining and aggregation of record sets. In this phase, user queries undergo reformulation [DHY09, HRO06]. Queries posed over the global schema are translated into a form executable by local schemas, and reflect the semantic mappings derived earlier.

vi Closely related to querying results is the process of *integrating instances*. This is where semantically similar records are identified. Usually, text-matching techniques (for example, Edit distance, Smith-Waterman distance, Soundex) are used (in combination more than singularly) to identify potentially similar records [BMCF03]. Despite that, false positives can still occur. To address that, *ontologies* [Gru93] may be used together with the selected matching techniques. This appears promising, if each domain has a comprehensive set of ontologies. Instance integration in a dataspace has the capability to operate incrementally. As changes happen in the domain from time to time, a dataspace integration strategy incrementally accepts additional records, any updates to existing records, and revisions to the domain's logic. As a result, a grouped set of similar records is produced. The latest proposal [WGM14] on incrementally integrating instances has exploited the reusing of earlier-produced integration results in order to remove the need for re-integration of the entire extent of data.

vii *Data fusion* in dataspaces has to incrementally transform similar records into *"a single, consistent and clean representation"* of a real world instance. It is the final phase before results are available for user viewing. Recent effort [HdACC13] involves fusing data in an incremental manner, using a special form of provenance information kept as a series of operations, taking on the idea of logging activities in manual data curation. This information does not hold a history of user actions. Instead, it contains the data's original values carefully coordinated with their local sources.

viii *Improving integration* typically builds on feedback, for example from users or crowds. User feedback, in dataspaces, can be provided either explicitly or implicitly. When using the former approach, users deliberately validate *artefacts* of a dataspace, such as matchings, mappings, global schema(s), reformulated queries, results of the queries or the ranking of the query results. In contrast,

use of the latter approach means exploiting *by-products* of a user's interaction with the dataspace. Those that potentially could be considered are logs of submitted queries, sequences of navigations and specific data points that have undergone further drilling-down. Additionally, the effects of improvements can be directly on the artefact on which feedback was obtained or on other indirect artefacts.

1.2.3 Challenges

Like most data integration efforts, dataspaces too has its challenges. In this subsection, we discuss some of the many challenges facing dataspaces. Our discussion is framed around the logical components of a dataspace which appear throughout its life.

Data Sources and Semantic Relationships

In dataspace, there can be a multitude of available data sources, each with different lifespans, depending on the continuous commitment given by their creators [Lev98]. A dataspace has to cope with a changeable set of data sources, which eventually would affect the global schema's existing definition and the extent of any integration performed thus far.

Data sources that participate in a dataspace (*a.k.a. participants*) need to be discovered [FHM05]. We shall use the terms *participants* and *data sources* interchangeably throughout this thesis. In big organisations, especially, the number of data sources owned can be overwhelming. Discovery is important so that relationships between them can be formed, and any existing relationships can be improved.

Each source in a dataspace, commonly, adopts one data model. A dataspace is expected to support an array of data model types [FHM05]. The more types of model a dataspace can support, the more participants it can accommodate, hence, becoming more comprehensive to users. To support a variety of data models can be a tricky business. Some issues that need to be considered are the different query languages used with each model and the degree of structure inherent in a data model.

Relationships between participants require discovery [FHM05]. This involves matching source schemas and forming mappings between a global schema and a set of source schemas [HBF+09]. Automatic generation of these relationships is favourable, primarily because it does not engage users with the tedious job

of identifying and forming relationships. In return, it creates the presence of uncertainty about the validity of the generated relationships.

When automatic means are used for generation of mappings, most of the time, this results in a substantial number of records retrievable from the heterogeneous data sources as a result of overlapping candidate mappings. Unfortunately, this also includes false positive records. These records are uninteresting to users because they do not match the user's expectations. This *flooding* of records can lead to wasteful compute cycles when instances are being integrated. Mitigating this flood can be a challenge.

Instance integration addresses the identification of a real world object from its syntactic representation in multiple data sources. The challenge is these representations are commonly unique to their data sources of origin, and thus commonly display differences in relation to the terms used, data format and stored information. This difference is widely known as the *heterogeneity problem*.

More often than not, similar records from multiple sources can carry inconsistent values when they are compared against one another. In this condition, they must be transformed into a single, consistent, useful version [BN09] that agrees with the real world object being represented before being presented to users. This is better known as *data fusion*. A primary concern to be considered during fusion would be to decide, based on some ulterior information, as to which value of the records is more relevant. Some information about provenance can be helpful.

Queries

Query languages vary in expressiveness and processing capabilities. Queries are generally tied to their underlying data models which vary across data sources. Unless explicitly specified, users typically expect the query to consider all relevant data in the dataspace, regardless of the data model in which it is stored or the schema along which it is organised. Franklin [FHM05] categorises data models into two extremes denoted by *weak* and *strong* characteristics. A weak data model has a loose structure, for example, bag-of-words data model, while a strong data model is tightly structured, for example, relational data model. Queries posed over a dataspace's global schema must be reformulated. It is challenging to reformulate the complex query language of a strong data model to be used on a weak data model, and in contrast, to reformulate the simple query language of a weak data model for use on a strong model.

User Feedback

User feedback, either explicit or implicit in nature, relies on user's knowledge of objects and their interaction in the real world. Until now, there has been no formal framework for understanding and learning from user feedback in dataspaces. The need for a formal framework was highlighted by Halevy *et al.* [HFM06]. Halevy *et al.* insisted that the framework be equipped with three components: a definition of the problem that requires a human's attention; a formalism in the form of measurements of the elements involved in the problem (for example, similarity distance between two schematic matchings); and a space of possible solutions.

Users interaction activities with a dataspace can be a good source of information, especially about the relationships between sources in a dataspace [HFM06]. Methods should be developed to capture and interpret these activities. However, this is no easy task. The implicit nature of these activities requires extensive interpretation of the intent behind every sequence of the activities performed.

Although users are available to provide feedback in exchange for better integration quality, however, this can be expensive. The time users spent to validate a dataspace artefact can be used for some higher cognitive tasks that cannot be performed by a program. Hence, methods should be devised to examine existing semantic relationships, identify artefacts that could assist in their improvement, and ask users to specifically validate these artefacts. The aim is to fully utilise the time users are willing to spare.

In dataspaces users are given an attractive incentive whenever they participate in the resolution of instances. The more domain knowledge, in the form of feedback that they supply, the better the quality of integration that they get to gain. However, users do not specify feedback to give clues to the dataspace on the different categories of records that exist in a data set of interest, instead, they simply specify what they know about the similarity of a specific pair of records from the data set. The challenge is to infer knowledge from such feedback to improve the integration.

1.3 Research Questions and Hypotheses

RQ1 How can we design an approach that identifies instances which are relevant to a user's information need in a dataspace?

RH1a Identification of instances that are relevant to a user's information need in dataspace is best viewed as a ranking of schema mappings

problem.

RH1b A user's information need can be known from terms found in a query's conditional clause.

RH1c Semantic mappings can be ranked based on their relevance to a user's information need specified by the terms found in queries.

RH1d The rarity of terms found in a semantic mapping's extent can indicate how relevant the mapping is to a user's information need.

RQ2 How much query logs is needed to produce stable rankings?

RQ3 Would the proposed ranking technique be able to track query patterns that are data source-specific i.e., *skewed*?

RH3a Some data sources tend to amass a large collection of terms, forming patterns, that may not be frequently used by other data sources.

RQ4 How can we devise a strategy to integrate instances from different large data sources in a dataspace?

RH4a We cast instance integration of large data sources in a dataspace as the problem of incremental clustering.

RH4b User possess valuable domain knowledge (explicit feedback) that can be useful to continuously improve the integration.

RH4c Due to the unique characteristics of most datasets, tuning the configuration parameters of the underlying incremental clustering algorithm is necessary.

RH4d A useful way to handle the different information requirements from users is to guide an objective function to reflect user's understanding of the domain.

RQ5 Would relying entirely on domain information supplied by user feedback be adequate to improve instance integration in the occurrence where integration of instances is considered to be a black box?

RQ6 How effective is our approach(es) in integrating instances at different levels of user feedback?

RQ7 For scalability reasons, how can we achieve an integration quality that is comparable to the integration achieved with the use of a full set of data but while using only a fraction of the data set?

RH7a Pruning a data set down to have only informative records remaining will not compromise the produced integration quality.

RQ8 Would our pruning strategy be able to produce a higher degree of integration than when integration is done on the full data set?

RQ9 Would the time to complete an integration process be faster with a pruned data set produced based on hypothesis RH7a than when the complete data set is used?

1.4 Aim, Objectives and Contributions

The aim of this thesis is to investigate the use of pay-as-you go approaches for instance level data integration.

1.4.1 Objectives

O1 To design an approach that ranks mappings based on their relevance to a user's information needs.

O2 To devise a strategy to integrate instances from different large data sources, and that takes advantage of knowledge from users, in the form of feedback, about the domain of interest.

O3 To design a technique which could integrate large amounts of data by using only a fraction of the complete data set when testing the fitness of candidates within the evolutionary search but without compromising the integration quality achieved when the approach in O2 is used. The reason is our instance integration strategy can be quite expensive. Repeated clustering of the complete data set is needed during the evolutionary search. With large data sets, this is challenging.

1.4.2 Contributions

- An approach that identifies instances relevant to a user's information need, by using terms commonly used to describe the instances, which can be found in query logs. We view this as a problem of ranking mappings, since in a dataspace, records are retrievable through generated mappings. This approach satisfies RQ1 of Section 1.3. To our knowledge, we are the first to tackle this issue in a pay-as-you-go data integration setting.

- Two empirical evaluations of our approach. The first evaluation seeks to answer how quickly stable rankings can be produced over a sequence of different log sizes (RQ2). Our results show that our proposed approach can produce stable rankings for encouragingly small log sizes.

  The second evaluation investigates how the rankings track query patterns that are *skewed* towards specific sources (RQ3). The ability of the ranking to reflect skewness allows for the identification of data sources that contribute to *"useful"* mappings, translating to which data sources that store instances needed by users. Our evaluation was conducted on two real-life life science data sets. From our evaluation, the generated ranking responded satisfactorily to the *level of skew* inherent in the query logs.

- A strategy which works on the concept of clustering to group similar records together in relation to their syntactic data value, using feedback to identify which records have been correctly clustered. Weights are applied, indicating the discriminative influence which an attribute has on the data set. Additionally, special parameters are used to configure the clustering algorithm. In order to determine a good set of weights and configuration parameters, we turn to evolutionary search and the use of user feedback to evaluate the fitness of alternative clusterings. This strategy answers RQ4.

- An empirical evaluation, comparing different variants of our proposed strategy against a baseline. We tested our strategy on three public data sets commonly used by the entity resolution community. The variants explore optimisation by weights only, weights and parameters, and post optimisation score change. We also tested when no optimisation is conducted, but instead similar scores of record pairs are directly changed based on user's feedback. We are interested to know the relative performance of these variants (RQ6). From our results, we can see that optimisation of attribute weights and parameters of the clustering algorithm generate fitter clusters than clustering alone with manual parameters. However, directly changing similarity scores does not give better results. Nevertheless, optimising weights and parameters of the clustering algorithm produces fittest clusters when compared to the other variants.

- A technique that, based on user's feedback, selects records which has feedback specified on it. In other words this technique prunes the complete data set by choosing only the records that have been validated by a user. This

technique satisfies RQ7. Since we have found that optimising weights and parameters can give fit clusters, we used that particular variant with our pruning technique.

- A comparison between clusters generated from the pruning algorithm and those of the complete data set (RQ8). Our evaluation shows that, on most of the data sets, our proposed pruner produced comparably fit clusters when larger feedback amount is used.
- A comparison of run time between pruned data sets and their complete counterparts. We can observe that with the pruned data sets the run is faster by several orders of magnitude than when the full data sets are used.

1.5 Thesis Organisation

The organisation of this thesis is as follows. Covered in Chapter 2 is our technical descriptions on mappings, instance integration and user feedback, introducing readers with some general background to our research. We present a ranking strategy that places mappings in a precedence of relevancy to user information need in Chapter 3. In the same chapter, we report two empirical evaluations. Moving on to our next proposal is Chapter 4 which discusses in detail the five different variants we proposed to conduct integration of instances, followed with a report on the empirical evaluations conducted on them. Faced with large data sets, our most promising instance integration proposal needs to be scaled accordingly, hence, we suggest a data set pruning strategy in Chapter 5. This thesis is completed with a review of our contributions and the future direction, both illustrated in Chapter 6.

Chapter 2

Technical Context

In this chapter, we present the concepts and terms of relevance to this thesis. We provide descriptions for: the *heterogeneity problems* that are faced by all data integration proposals; *schematic matchings* and *semantic mappings* used in the effort to create correspondences between heterogeneous data sources; and the *instance integration* process that identifies similar records from the pool of records made available through the discovered correspondences. Essential to dataspaces, we describe the different types of *user feedback*, and we touch upon a newly-introduced type of feedback, *crowdsourcing*. Also, we explain how our proposals fit into a general dataspace architecture and describe their interplay with common components of a dataspace. As a general framework for dataspace, we have chosen DSToolkit [HBM+12] because of its comprehensiveness. Then, we discuss existing proposals for dataspace, specifically those that deal with the problems we are addressing. For our first problem of *instance integration in a pay-as-you-go setting*, we try to answer the question *"do present dataspaces support instance integration?"* If they do, *"what is the method that was used to perform integration?"* Furthermore, *"what is the role of the user in the integration?"*. In contrast, with our second problem, *the identification of instances relevant to a user's information need in dataspaces*, at the time of writing this thesis, we are the first to tackle this problem. Addressing data relevance is not new and proposals can be found in areas of information retrieval as well as data management. In information retrieval, where documents are often the concern, a relevant document is one that is perceived by users to contain valuable information in terms of their individual information need [MRS08]. A similar view can be found in data management where data relevance is considered to be hard to evaluate objectively, owing to its great dependency on how the data is used. A data item may be relevant in

one task, but not in another.

2.1 Heterogeneity Problem

When heterogeneous, independently-constructed data sources are integrated, they can give rise to the *heterogeneity problem*. Sources were constructed to fulfil the information need of their own stakeholders. Their model, structure and contents can present different degrees of uniqueness and similarity. Identifying similarity in integration systems is not an easy task. The problem of heterogeneity can occur at different phases of an integration system and on different aspects related to it, from the point of communicating with a data source to the values of the source's contents. We describe heterogeneities [GMUW08] in this section.

- *Communication heterogeneity*: there are many choices of communication protocols, such as HTTP for the web, remote procedure calls and FTP for remote accesses. Given these, retrieving data in an integration system can require managing a repository which spells out each source's acceptable protocols and the necessary software.

- *Query-language heterogeneity*: exists as a result of a variety of data source types (e.g., XML, relational, object, Excel spreadsheet) used by integration systems. Each language is tailored to handle different degrees of schema and structure, such as XQuery and SQL. Even if an integration system manages only relational databases, SQL comes in different *dialects*.

- *Schematic heterogeneity*: even if only relational databases are used for integration, the schemas applied can differ. Attributes may be combined or separated; concepts may be designed as a single relation or placed as an attribute; and there can be dissimilarity in the information covered.

- *Type heterogeneity*: *integer*, *boolean*, *character*, *string* and *date* are forms of data. For example, a serial number can be stored as either an *integer* or a *string*, depending on the perception of the database designer and the operations to be performed on it (e.g., *concatenation, append*).

- *Data heterogeneity*: data values describe the characteristics of a real world object. They can be stored in a string (e.g., BLACK, BL) or represented by an integer, like the number 1. In another data source, the string BL or the number 1, may correspond to *blue*.

2.2 Schema Matchings

A *matching* is created when the elements of a pair of local schemas are linked together, and the link is postulated to represent a semantic relationship. Manually constructing matchings is expensive, due to the fact that it is exposed to human error, consumes substantial time and is laborious. Many efforts toward improvement incline toward providing automatic support. Rahm *et al.* [RB01] emphasised the importance of having a *"generic, customisable implementation"* of a Match operator which is agnostic of the underlying application area to realise this vision for automation. An automated Match can be realised in many ways; comprehensive surveys of automatic schema matching include [RB01, SE05]. We describe in the following subsections the classification by Rahm *et al.* [RB01] based on its higher number of citations.

2.2.1 Schema-level Matching

Schema information such as name, description, data type and constraints (e.g., integrity and referential constraints); relationships between schema elements, such as *part-of* and *is-a*; and schema structures, such as relational and XML models are used by schema-level matching to identify matches between source and target schemas. Two forms of matching techniques dominate this class of matching approaches: *element-level* and *structure-level*. Their difference is characterised by the granularity and cardinality of Match.

- *Element-level matching techniques*: advocate for a single element matching and limits to the cardinalities of *1:1*, *1:n* or *n:1*. One major matching technique involving elements is *string-based techniques* which manipulate strings in elements without ignoring their lexical structures. *Lexical techniques* are another major element-level matching technique. Aimed at preprocessing strings prior to applying string-based techniques, lexical techniques exploit natural language processing methods to parse the string elements into individual tokens, root words and words known to denote real-world concepts. Besides string-based and lexical techniques, *semantic-based techniques* are also deployed at the element level. The purpose is to take a step further in estimating similarity that is by considering the *meaning* behind the sequence of characters representing each element. These techniques depend on the semantic associations of two strings such as synonymy, hyponymy and hypernymy.

- *Structure-level matching techniques*: support matching of a combination of elements that show some structural relation, and handle cardinalities of *m:n*. A matching technique falls into one of two groups of strategies. The first is based on *scope* which indicates the extent of elements in a schema over which matching is performed. [MGMR02, TC07, DR07, TLL+06, MBR01] permit all elements of a schema to be included in the matching process, while [HQC08, XE06] restrict the inclusion to specifically chosen ones. The second is *neighbourhood* that an element resides in, which holds the power to influence the matched elements based on the special position that element has against the matched elements. Works involving neighbours include [DR07, TLL+06, MBR01, ASS09, MGMR02].

2.2.2 Instance-level Matching

An instance or record provides valuable insight into the contents and meaning of a schema elements. This is especially true when there is a shortfall in useful schema information or there is a complete absence of any schema. Such high-level information can help reveal incorrect interpretations of schema even when ample schema information is obtainable. Unlike the earlier described approaches which suit both schema-level matching and instance-level matching, the following approaches are especially for instance-level matching.

- *Linguistic characterisation* [RB01]: preferable when text elements are presented. This approach works by separating keywords and themes having a certain amount of frequencies of occurrence of words and word combinations.
- *Constraint-based characterisation* [RB01]: used with numerical and string elements. It includes ranges of numerical values or character patterns.

2.3 Mappings

Mappings describe the semantic associations between a global schema and a set of local schemas, and are an essential component in an integration system. Fundamentally, a mapping is a database *view* or also known as a *virtual relation*. A virtual relation does not store any data, but contains a description of a source which specifies its contents, attributes, constraints, completeness and reliability, and query processing capabilities [Lev98]. Mapping is the cornerstone of reformulating user queries. User queries posed over a global schema are translated

into a query language comprehensible by each source schema, making use of any related mappings.

2.3.1 Mapping Definition

Three most widely investigated approaches to defining views have been proposed for traditional data integration systems [Len02]. They are *Local-As-View (LAV)* [DGL00, FW97], *Global-As-View (GAV)* [GMPQ$^+$97, ACPS96, Ull97] and *Global-and-Local-As-View (GLAV)* [FLM99, HIST03].

When LAV is used, a mapping is produced for every element in the local schema in the form of a view to the global schema. LAV is suitable if changes to the global schema are infrequent, while addition of new sources is relatively common. Introduction of new (or changed) sources basically requires the creation (or adjustment) of a view, with no changes needed to the other mappings. Unfortunately, rewriting a user query in LAV is not a straight-forward process. An equivalence has to be sought between the views.

Conversely, in GAV, mappings are generated for each element of the global schema as a view over some local schemas. Such mappings present clear information on how data is to be retrieved when elements of the global schema are evaluated. In cases where new sources are not frequently added, GAV may be a suitable choice. The reason is that adding of new sources or changes to currently registered ones may require changes to many other mappings. Hence, GAV is not scalable for large applications.

Offering the best of both worlds is GLAV. Like LAV, GLAV offers independence of a global schema, the ability to manage new sources and the complexity to reformulate queries [XE04]. On the other hand, better than LAV and GAV which use a restricted form of first-order logical sentences, GLAV uses flexible first-order sentences, allowing a view over source relations to be a view over global relations in source descriptions. Hence, with GLAV, data is derivable from views over source relations, and it allows conjunctions of global relations.

2.3.2 Generation of Mappings

The activity of mapping generation is the consequence of substantial human effort [HRO06]. Experts in the domain area and database must work together to construct mappings that comply with the application's requirements. Since the generated mappings can accurately fulfil user needs, they can be expected

to be of high quality, but in return, startup time is increased. An ideal condition is the (semi-)automatic generation of mappings, where users do not need to take part in the process or need not be given a central role. (Semi-)automatic means, unfortunately, can be complicated and inherently exhibit *uncertainty* in the correctness of the mapping. For positive results, many research proposals [FHH+09, BM07, PB08] turn to full schema definitions (e.g., referential integrity) or external resources (e.g., rich specification for the relationships of schemas) for support. This can be impractical as these components may not always be fully available, can be unclear or not well-defined. For example, Clio [FHH+09] takes as input referential constraints and any relational or nesting structures in order to form views as conjunctive queries i.e., *join* or *union* of schema elements. Similarly, Model Management 2.0 [BM07] requires the availability of primary and foreign keys as *joins*, while many rely on common elements found in local schemas; an example is Pottinger [PB08]. Asking for a user's validation can be helpful, however, it is only a handful of validations that a user should be appropriately asked to offer.

Apparently, (semi-)automatic view generation is faster and cheaper than manual construction of views. Less apparent is the multiple, overlapping mappings that can be produced. As such, mappings may not accurately meet the needs of users and hence may require several refinement processes [BPE+10].

2.4 Instance Integration

Instance integration can be described using many terms: record linkage [FS69], data deduplication [EIV07], entity resolution [BGMK+06, KTR10], identity identification, merge-purge. All these refer to the task of identifying records in multiple databases that describe a common object in reality. This is no trivial task. As we have described earlier, the different types of heterogeneity problems that occur at every turn of a dataspace's life, and appear in multiple aspects, consequently complicate instance integration. The naïve approach to instance integration is pairwise comparison of records; however, this will give rise to the $O(n^2)$ problem where n is the number of records and the cost of integration is $O(n^2)$ number of comparisons. As a result, instance integration can be costly, where it can take hours or even days to complete, depending on the size of the data set. Typically, to improve efficiency means to decrease the search space. In this section, we provide an example of instance integration. We further discuss techniques

used to address the two essential issues in instance integration: effectiveness and efficiency.

2.4.1 Improving Effectiveness

Effectiveness involves the level of accuracy that can be achieved when records from different data sources are identified to represent the same real world object. Ideally, accuracy is measured by some ground truth, unfortunately, this can be difficult to get. In reality, the ground truth may not be available or is unknown at that point in time. The cornerstone of achieving effectiveness is in identifying similarity between two records. Techniques to find similarity, in general, compare values that describe real world objects. The intuition is that records representing the same real world object can be expected to have the same description. This is not always true, hence, additional information can be provided by using ontologies. A common description of an ontology is *"a formal, explicit specification of a shared conceptualisation"* [Gru93]. Effort to leverage ontologies in data integration is present across domains. They include [SAR+07, FDO+01, CSC04, BDPH06, CT09, BL04]. Unlike an ontology which focuses on the semantics of concepts, our work relates concepts using textual syntax, hence, ontology can be seen as complementary to our work. Elmagarmid [EIV07] suggested two classes for similarity comparison techniques.

- *Compares individual fields for similar contents*, where each field has different types of data that are afflicted by specific types of errors. Acting upon these errors would require distinct similarity functions, capable of handling such specificity. We describe here some of the available functions, narrowed to string-based metrics; and touch upon numeric-based metrics which are less established.

  - *Character-based similarity metrics*: address typographical errors. A widely-used metric, *Edit Distance* or also known as Levenshtein distance [Lev66], determines similarity by calculating the total cost of using edit operations to transform one string to another. The smaller the cost, the more similar the strings are since not many edits are required. A cost of 1 is assigned to every operation performed. They involve *including* a character into the string, *deleting* a character from a string, and *replacing* one character with another. Edit distance is effective in capturing typographical errors, but not other types of incorrectness.

- *Token-based similarity metrics*: in many conditions, typographical differences lead to the repositioning of words. Such changes cannot be identified by simply dealing with characters, but require the participation of tokens. One form of token-based metric is *Atomic Strings* [ME+96]. The basic working unit of this metric is a series of alphanumeric characters separated by punctuation characters, or known as atomic strings. When two atomic strings are equal or when one atomic string acts as a prefix to another, they are considered to be a *match*. Similarity is achieved when the number of matching atomic strings is divided by the average number of atomic strings. The higher the calculated value, the more similar the attributes are.

- *Phonetic similarity metrics*: besides having similarity in characters and tokens, phonetically similar words such as *Kageonne* and *Cajun* can also exist in database records and could be identified by special-purpose metrics. A regularly-used phonetic-based metric is *Soundex* [Rus18]. Fundamentally, Soundex uses consonants as an element for discrimination, where similarly-sounding consonants are assigned a common numerical code. Therefore, both *Robert* and *Rupert* will return the same code of *R163*. This works well with Caucasian names, but is less effective with East Asian names, owing to their extensive use of vowels [New67].

- *Numeric similarity metrics*: is reported [EIV07] to being operated upon as strings, using the above-mentioned metrics or basic range-based queries, in order to find similarity. A potential course for research is to take into account the distribution and type of the numeric data in determining similarity; or use of cosine similarity and any of its variants [KMS04]. However, there are no further advances on this subject matter found.

- *Compares whole record*. In reality, records are not compared by just a single attribute. Multiple attributes are needed to determine if a record pair is similar. Present proposals can be grouped into the following categories based on the absence or presence of user participation.

  - *Proposals with user involvement*: Strategies under this category have

been applied with the involvement of a user. Validating (i.e., specifying if a record pair is similar or not) and characterising data (i.e., labelling training data based on the ground truth) are the primary roles given. Strategies that we describe here include probabilistic-based, (semi-)supervised learning and active learning-based.

In essence, *probabilistic-based* strategies build upon the notion that instance integration is a Bayesian inference problem, and these strategies aim to categorise a record pair as being $< \alpha, \beta > \in M$ or $< \alpha, \beta > \in U$, where M is a set of duplicates and U is a set of non-duplicates [EIV07]. Bayes theorem is used to calculate the likelihood which indicates the probable set that record pair $< \alpha, \beta >$ should belong to. Probabilistic models [FS69, New67] are driven by two aims: firstly, to minimise the probability of incorrectly classifying a record pair. Secondly, on the premise that misclassification may have different consequences, the notion of assigning a cost to minimise error is introduced. The set membership probability can be calculated using training data of prelabelled record pairs.

A set of training data which is prelabelled with the ground truth is a prerequisite if a *supervised learning* technique is to be used. Such techniques involve learning probabilistic models, deterministic classifiers and characterising duplicates [CCMO11]. Unfortunately, this type of learning relies heavily on the availability of a comprehensive collection of training data, presented with diverse cases of duplicates. The more various the cases, the more accurately the classifiers can be tuned. In practice, these conditions rarely exist [CCMO11].

An alternative approach (*active learning-based technique*) to supervised learning is to actively select subsets of unlabelled record pairs that would give *"highest information gain"* [SB02] when labelled. These special record pairs are identified as the ones that possess *uncertainty* [SB02]. Given the record pairs in question, users are asked to validate their *similarity* status. The labelled data are then used to re-train the learner.

– *Proposals without user involvement*: In some cases where training data

and user participation are not available, both supervised and active learning will not be useful. Techniques under this category do not require any user validation or characterisation, instead they rely on metrics to help compare a record pair's similarity.

Distance-based techniques treat a record as a single long string and find similar records by using one or more distance-based metrics which we have described earlier when comparing individual fields for similar contents [EIV07]. A record pair is considered similar if it passes a *similarity threshold*. A disadvantage is that attributes are considered as non-discriminative, hence, their embedded information (e.g., data type) is not used during identification of similar records. Another disadvantage is the problem of determining the correct threshold. If training data is used, finding an appropriate threshold value is possible. But, this would beat the purpose of offering a "no-user-involvement" strategy.

Another strategy in this category is *unsupervised learning*. Typically, manual labelling of record pairs can be bypassed with the use of clustering algorithms. Clustering algorithms assume that similar records correspond to the same class. Clustering can be presented as being hierarchical or partitional. Hierarchical clustering *"recursively finds nested clusters"* [Jai10] by agglomerative or divisive means, depicted as a dendogram. Agglomerative clustering places every data within a unique cluster and progressively merges the clusters where the containing data are similar. Divisive clustering takes on an opposite approach, starting with a single cluster of all data and repeatedly dividing each cluster based on the dissimilarity of its data. Conversely, partitional clustering finds all clusters at once as partitions of data [Jai10].

So far, there is no single technique that can be the silver bullet, since one may handle a single dimension of similarity. Hence, a combination of techniques may be used to achieve better effectiveness.

2.4.2 Improving Efficiency

Efficiency is related to how quickly the detection of common representations across a set of data sources can be achieved. The naïve approach of comparing every record in a table to every other of another table would require $|Table1| \times |Table2|$ number of comparisons (with $|.|$ denoting the number of records in a table), and this can prove to be prohibitively costly [EIV07]. At a finer granularity, comparisons may also be done between attributes of different tables [EIV07]. Owing to the potentially significant number of attributes any tables can have, the total number of comparisons that must be performed can be large. Most techniques that are concerned with efficiency aim at reducing the number of comparisons needed between records, in the effort to reduce the time of completion. These proposals can be grouped into the following strategies [EIV07].

Blocking aims to reduce the number of record comparisons by limiting comparisons only to records within the same *block* [WMK+09, PI12]. Blocks are non overlapping. Fundamentally, a block is a group of records which share the same *key*, and this *key* represents a common criterion. A key can be a single record attribute, a concatenation of values from multiple attributes or a calculated product of a function (e.g., a hash function). A *good* blocking key is one that can group *similar* values together. Similarity can be specified by some measure on a particular characteristic, for example, similar sounding words or similar looking words, which can be found using phonetic encoding functions such as Soundex, NYSIIS, Double-Metaphone [Chr06]. Blocking can greatly improve efficiency. However, it can also result in false negatives (correct results that are viewed as incorrect) due to the fact that blocking separates what is viewed to be dissimilar records from the very start, which is a result from the choice of similarity function used, or an error in the blocking step, placing records in the wrong block.

Sorted neighbourhood makes the assumption that similar records can be sorted according to some common characteristic. It comprises three steps: create key, sort records and merge. At the beginning, relevant attributes or parts of attributes are harnessed to *produce a sorting key*. Next, *records are sorted* into a list based on the key, where attributes that are more likely to discriminate records are placed earlier in the sorting key, specifying a given higher sorting preference. Finally, during *merging*, comparisons are performed between records that are grouped together in an invisible *box*, shifting down the sorted list. For example,

if the size of the box is n (where n=2) records, the box shifts to receive a fresh record, leaving the first one, and the pair of records in the box is compared. The efficiency of sorted neighbourhood would be disrupted if the generated sorting key does not place duplicates near one another. In cases like this, duplicates have a very small chance to be found.

Clustering and canopies. A cluster is a group or collection of records that shares some commonality. Instance-wise, a cluster is in particular a set of records that point to the same object in the space of reality. Clustering is a strategy devised on top of the concept that the duplicate relationship is transitive so if record α and record β are duplicates, any record that is a duplicate of β, for example σ, is also a duplicate of α. In concert with this perspective of transitivity, central to clustering is the identification of connected nodes in an undirected graph, where these nodes represent the transitive closure of the duplicate relationship. In basic clustering, for every cluster, a record is elected as a *representative*. Any comparison to find similarity is performed against this representative record. Hence, the total number of record comparisons can be greatly reduced. The success of this approach leans upon how well a representative can deputise for its cluster. Like blocking, basic clustering adopts hard partitioning, where no record can belong to more than one cluster and like blocking, false negatives may occur.

A proposal by McCallum *et al.* [MNU00] reduces the number of record comparisons by initially using a cheap approach to produce *overlapping* groups, namely *canopies*, and then applying a more expensive method(s) to improve on the results. The success of *canopies* depends on the presence of a fast, inexpensive method. Cohen *et al.* [CR02] use Term Frequency/Inverse Document Frequency (TF/IDF) with canopies to cheaply calculate the distance between record pairs, while Gravano *et al.* [GIJ+01] proposes q-gram as the costly distance measure. An experimental comparison conducted by Baxter *et al.* [BCC03] showed canopies to have higher completion time and improved quality than basic clustering.

Balancing between achieving effectiveness and efficiency is not straight-forward. This decision is influenced by factors such as the goal of an application, the scale of the data and the domain in which clustering is being used.

2.5 User Feedback

Items of feedback are annotations that a user provides [BPF+11]. Feedback carries a user's knowledge of the domain. Feedback can be *explicitly* acquired, by posing a question to a user or by letting a user freely annotate a set of presented artefacts, or *implicitly* collected by unobtrusive ways. Irrespectively, at present, feedback has been used for validation [MSD08] and learning [TJM+08] purposes, with the aim of improving the quality of integration [BPF+11].

Belhajjame [BPF+11] highlighted two common assumptions, concerning user feedback, that are impractical and can stunt any potential gains. The first assumption is that, in a data integration system, only a single user is available, or all users are interested in the same requirement. In reality, requirements may differ by user. Inconsistencies in user requirements can occur and can cascade to produce inconsistencies in feedback. The second assumption is that, user requirements remain persistently unchanged. With changes in an organisation's policy or due to government law, it is inevitable for user requirements to undergo change. Such changes may result in a contradiction between previous and later requirements, accordingly, impeding necessary improvements.

2.5.1 Explicit

Belhajjame *et al.* [BPF+11] presented a list of the various dataspace artefacts that have been used to tap into a user's knowledge in an explicit manner (Table 2.1). An explicit item of feedback is specified on artefacts of an integration system such as matchings [MSD08], mappings [JFH08], queries [CQCS10] or query results [ACM+08, BPE+10, CVDN09, TJM+08]. It can come from a *controlled vocabulary* or *domain ontology*. For example, it is not limited only to validating via 'true' or 'false' answers, but it can also be used to learn by re-ordering result tuples with 'before' or 'after' information. An open-ended feedback is where, given two attributes of a relation, users are asked to provide a set of relevant constraints.

A primary advantage of using explicit feedback is that guesswork on what the user wishes to convey is avoidable. The reason is in the deliberate manner by which the user provides the feedback; a single item of feedback on a specific artefact. This eliminates ambiguity and establishes confidence. Another advantage is that explicit feedback indirectly fosters awareness in users of their contribution and role in the integration process. Such awareness can create a greater sense of

Table 2.1: Explicit feedbacks with their corresponding artefacts in current proposals. [BPF+11]

Proposal	**Artefacts on which feedback is given**	**Set of terms used for annotating artefacts**
Alexe *et al.* [ACM+08]	an instance of a given schema and the instance obtained by its transformation into another schema	{'yes', 'no'} used to comment on schema transformation
Belhajjame *et al.* [BPE+10]	a result tuple	{'true positive', 'false positive', 'false negative'}
	an attribute and its value	{'true positive', 'false positive', 'false negative'}
Cao *et al.* [CQCS10]	a candidate query	{'true positive', 'false positive'}
	a pair of candidate queries	{'before', 'after'} used for ordering queries
Chai *et al.* [CVDN09]	a view result tuple	{'insert', 'delete', 'update'}
Jeffery *et al.* [JFH08]	a mapping	{'true positive', 'false positive'}
McCann *et al.* [MSD08]	a relation attribute	set of attribute data types
	two attributes of a given relation	set of constraints
	a match	{'true positive', 'false positive'}
Talukdar *et al.* [TJM+08]	a result tuple	{'true positive', 'false positive'}
	a pair of result tuples	{'before', 'after'} used for ordering results

ownership, and potentially encourages users to *pay* more, hence, speeding up the maturity of the system.

Nevertheless, explicit feedback is limited in amount. Every given item of feedback bears the cost of the time that has to be spent in its provision. For this reason, asking for feedback should only be when necessary. This leads to the question of what feedback and on which artefact would generate the best outcome with the smallest cost. Asking for the right feedback is as essential as asking it on the right artefact. Serious thinking has to be put into what information would we like to solicit from users. Also, which artefact would deliver greater impact on the overall quality of an integration if the opportunity is given to engage with the knowledge of human experts.

Another form of feedback has been explored, namely implicit feedback, which we address in Section 2.5.2. It is by no means intended to replace explicit feedback, instead it is complementary by providing additional information at no cost.

2.5.2 Implicit

Implicit feedback is information that can be collected in an unobtrusive manner by observing a user's regular behaviour and interaction with a system [KT03]. Implicit feedback has been argued to be generally less accurate than explicit feedback [Nic97], but others have argued that its abundance and *no-payment* policy for users is an interesting alternative to explicit feedback [KT03, OK01]. Due to

the absence of classification on implicit feedback in the literature, we propose a classification that extends Belhajjame's [BPF+11], but presents properties that are relevant to implicit feedback (Table 2.2).

Table 2.2: Objects used to imply user feedback.

Proposal	**Objects used to imply feedback**	**Part of the object used**
Maskat *et al.* [MPE12]	query logs	terms in a query's condition clause
Yakout *et al.* [YEE+10]	transaction logs	transaction data e.g., products bought
Iofciu *et al.* [IFAB11]	set of tags	the tag itself
Tran *et al.* [TMC12]	list of entity descriptions	the description itself

Implicit feedback is inferred from existing artefacts/objects of an integration system. Examples include query logs [MPE12], transaction logs [YEE+10], a set of tagging activities [IFAB11] and a list of RDF entity descriptions [TMC12]. An object can contribute information by two means. The first mean is from the information that it holds, e.g., a term used by a query's condition clause or the occurrence of specific data in a transaction. The second mean is through the object itself, e.g., the frequency of tagging, and the merging of entity descriptions in RDF.

The advantages of using implicit feedback are *a*) it requires no additional activities to be performed by the user, hence, no extra costs; *b*) the user is unaware of any feedback-gathering exercises – in other words, it is unobtrusive; and *c*) it is an untapped wealth of available information. Unfortunately, implicit feedback can be imprecise, considering it is not a clear, definitive user-given answer. Using objects that are *by-products* of a user's interaction with an integration system relies substantially on a high degree of effort to infer a conclusion from the interactions.

In our work we use explicit feedback that provides information if a pair of records are (or are not) representing the same real world object inferred. We also use implicit feedback in the form of query logs to infer the information requirement of users.

In the next subsection, we briefly discuss a recently introduced form of explicit user feedback that is being explored in integration systems, namely *crowdsourcing*.

2.5.3 Crowdsourcing

Crowdsourcing is *"the process of obtaining needed services, ideas, or content by soliciting contributions from a large group of people, and especially from an*

online community, rather than from traditional employees or suppliers"[1]. We view crowdsourcing as a mass form of *explicit feedback* supplied by experts with interest either in the contents of the information system (*stakeholder*) or in the monetary incentives (*non-stakeholder*) offered. A recent survey [CFM+14] analyses proposals that chose to engage the power of the crowd for different data management-related tasks, such as *confirming* mappings [PF13], matchings [MSD08, ZCJC13, HTMA13] and classifications [SLB12]; or *providing values* for filtering [PGMP+12] and data mining [AGMS13]. In Table 2.3 we only list proposals from the survey which have instance integration-related crowd tasks, and have left the rest for interested readers.

Table 2.3: Crowdsourcing proposals regarding instance integration.

Proposal	**Crowd members**	**Crowd tasks**
ZenCrowd [DDCM13]	non-stakeholders	confirm value
Silk [IB13]	stakeholders	confirm value
Whang *et al.* [WLGM13]	-	confirm value
CrowdER [WKFF12]	non-stakeholders	confirm value

Owing to the enlisting of non-stakeholders and the offering of monetary incentives, crowdsourcing has specific issues to consider. Some of them are *a*) how critical is the subject to be worth having money offered to get an answer? *b*) since there are costs related, how do we get the most benefits from the crowd [DKMR13, WLGM13, JSD+13]? *c*) what is a good, acceptable size for a crowd such that there is confidence in the received consensus? *d*) do all crowd members have the same level of expertise and performance? If not, then how to identify and discriminate them [JGMP13]? *e*) how confident are we of the result of a crowd task, in terms of its accuracy [JGMP13, DKMR13, WLGM13]?

It is not surprising that the manner we expect a user to provide feedback in our setting resembles that of crowdsourcing — on records and explicit in nature. However, the incentives offered to entice feedback differ. With crowdsourcing, money is involved apart from the expected improvement in integration quality, whereas our users are promised only the latter. It is possible to view the type of feedback we requested to also be obtainable from crowds with the purpose of resolving the similarity between entities [WKFF12, WLGM13, WMGM, GDD+14, JSD+13, FKK+11].

[1] http://www.merriam-webster.com/dictionary/crowdsourcing

2.6 Dataspace Architecture

Several proposals for dataspace architectures [JFH08, BDG+07, FHM05, HBM+12, MCD+07, DS06] have been presented. In general, these proposals have the components essential to a dataspace, such as schema matcher, mapping or view generator, and query reformulator. For ease of understanding, we refer to DSToolkit as a reference architecture for a dataspace, functioning as a template. We revised the architecture to show the interplay of the operations for *ranking of mappings* and *instance integration* with the rest of the architecture, implying that our proposal could also be useful in other dataspaces simply by mapping components to DSToolkit. DSToolkit is the first dataspace management system built on top of model management principles. For this reason, DSToolkit offers easy management of heterogeneous schemas; automatic initialisation, maintenance and improvement; the ability to provide feedback on query result tuples; rewriting of queries that users posed over a global schema; and annotation and refinement of mappings. This makes DSToolkit a comprehensive framework to be generalised.

DSToolkit (Figure 2.1) consists of four layers: *Dataspace*, *Connectivity*, *Service* and *Presentation*. The *Dataspace* layer holds data about all relevant artefacts, i.e., schemas, matchings, mappings, user queries and user feedback, while the *Connectivity* layer describes access methods and is irrelevant to our work, hence it is not discussed here. The *Service* layer has all the actual functionality offered, such as model management operators, query processor, techniques for annotating, selecting and refining mappings with relation to a given user feedback; and the *Presentation* layer is the graphical interface for the interaction with users.

DSToolkit builds on model management [ABBG09] for many of its abilities to produce and manipulate dataspace artefacts. By extending from operators found in model management, DSToolkit is able to offer automatic initialisation, which includes matching of schemas (*Match*), inferring of correspondences (*InferCorrespondence*) and generating of views (*ViewGen*).

Post-initialisation in DSToolkit shows a tight connection between query expansion and mapping selection. In essence, any queries posed over one or more global schemas must be transformed into queries executable over each relevant data source. To do this, the posed queries are expanded by using unfolding techniques [Hal01], taking into account suitable mappings. Given the availability of multiple suitable mappings, and the absence of any knowledge of which mapping best satisfies a user's needs, the expansion can grow into sizeable proportions,

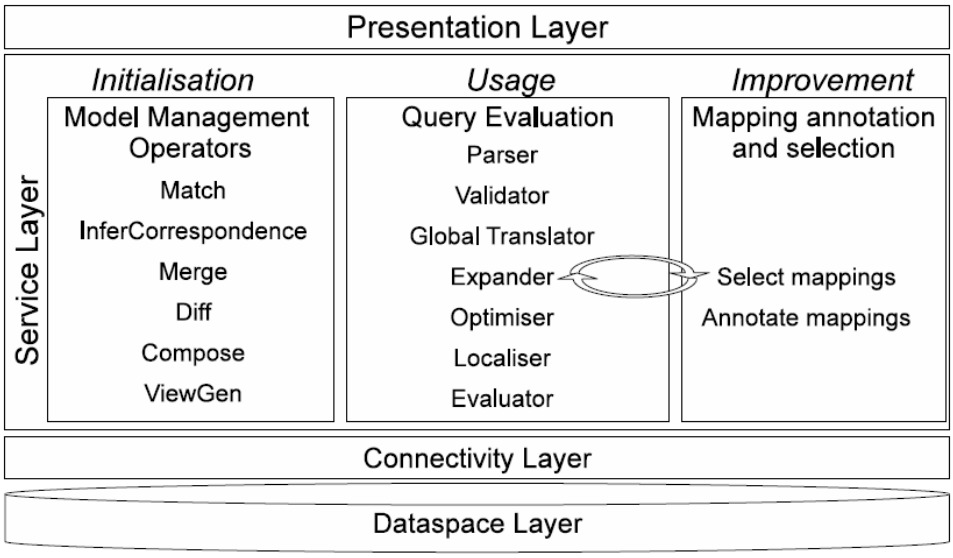


Figure 2.1: DSToolkit architecture [HBM+12].

since all suitable mappings are included. Such an elaborated expansion may result in many queries to be run on local sources. Some of these queries may return unwanted results, indicating unsuitable combination of mappings. DSToolkit manages this by inviting users to give feedback, in the form of validation, on the result tuples returned by such mappings. This then annotates the respective mappings based on how well their returned tuples meet user requirements. The gathered feedback helps in the next iteration of mapping selection decision.

We revise DSToolkit's original architecture (Figure 2.2) to include *ranking of mappings* and *instance integration*. Using automatic methods, mappings can be easily generated in large numbers. As we have discussed earlier, not all mappings are interesting to users. This may leave users with a large pool of result tuples, where potentially a considerable amount can be viewed as noise. We proposed to rank the mappings based on how close a mapping is to fulfilling a user's information need. Our assumption is that a user's information need can be guessed from terms that are found in a query's conditional clauses that a user has posed (refer RH1b of Section 1.3). Given M a set of candidate mappings, Q a set of query logs, the operation *rank mappings* takes M and Q and produces a list of ranked mappings M'. This *'improvement'* process can be repeated at specific intervals or as and when a condition is met.

When a user poses a query, the returned result tuples, in most cases, will contain duplicate records. To provide users with a single view of data, these duplicates must be resolved. Resolving instances requires the identification of similar

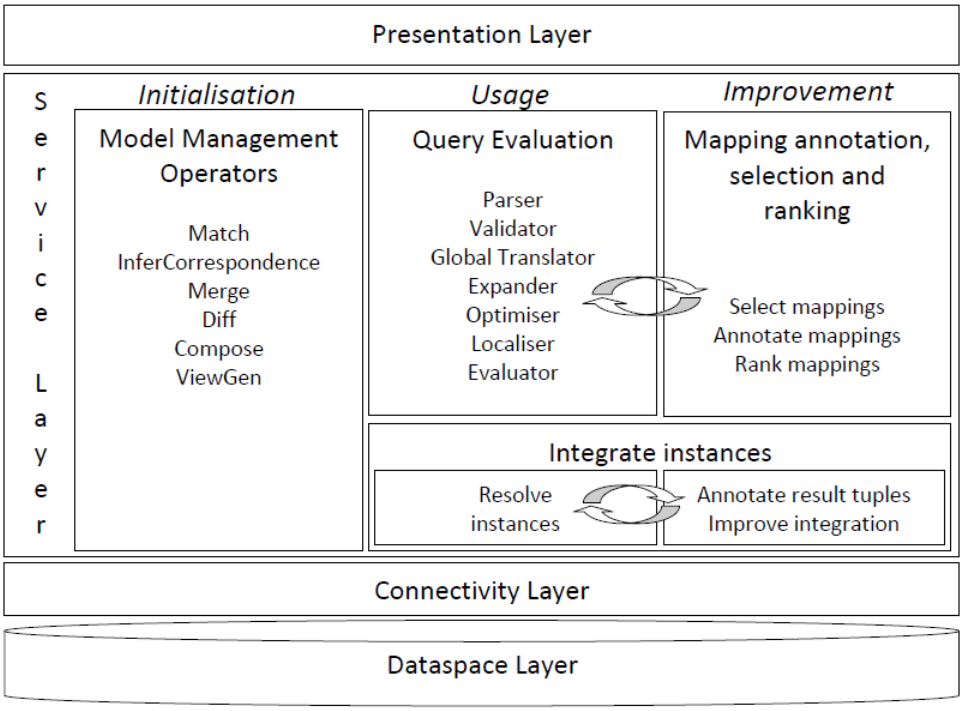


Figure 2.2: Revised DSToolkit architecture.

records which refer to the same object in the real world. Resolving instances can be initiated by a user after they have posed a query, or it could be automatically executed after every user query. The operation *resolve instances* receives R, a set of records that contains duplicates, and generates a set of resolved records R'. However, the degree of resolution is algorithm-dependent and may not comply with the actual truth. Although the degree of resolution should go hand-in-hand with the degree of compliance with the ground truth, the complete elimination of duplicates is near to impossible. Hence, we use the term "*partly-resolved* set of records" for R' to indicate the presence of *non-compliance* in the record set. We view this function as part of a *usage*-type of task, because resolution can occur without any improvements being made.

In dataspaces, giving feedback is not mandatory. However, users have been made aware that without any feedback from them, it is impossible to have improvements to the integration quality. An operation for this process would be to *annotate tuples* which has R', a set of partly-resolved records, and F, a set of feedback for each relevant record pair, as input and produces R_A, a set of annotated partly-resolved records. In order to transform R' to be more compliant with the ground truth, we proposed the *improve integration* operation which uses information found in R_A to *improve* R', and produce R_R, a set of further resolved

records and increasingly compliant to the ground truth. It is favourable to have a continuous cycle between the three operators so as to fulfil the pay-as-you-go principles of a dataspace.

2.7 Summary and Conclusions

In this chapter, we presented the technical context of a dataspace system, pertaining to the scope of our research. We started with a description of the many different forms of heterogeneity problems which compromise the quality of a result (Section 2.1). Then, we described the main artefacts of a dataspace (i.e., matching, mapping) in Section 2.2 and 2.3, discussed one of its important processes (i.e., instance integration) in Section 2.4 and illustrated a significant and useful device, the user feedback, in Section 2.5. The effectiveness and efficiency of integrating instances in dataspace are imperative to produce good and timely results and striking a balance is proven to be a challenge. These we have described in this chapter's Subsections 2.4.1 and 2.4.2. User feedback, be it implicitly recovered or explicitly requested, is a valuable component of dataspaces and one which we have investigated in this PhD. The large amount of typically readily available implicit feedback, in the form of query logs, has made it useful in finding relevant instances retrieved through mappings. We explain this proposed approach in Chapter 3. Feedback intentionally provided by a user on a dataspace's artefact is considered explicitly collected. With precise information from the user on their knowledge of the domain, we applied this for validating instance match. This is described in Chapter 4. We completed this chapter by illustrating where in a general dataspace architecture our proposals would reside and describe the role they play in relation to other components in a dataspace. We also described some existing work on dataspaces.

Chapter 3

Ranking (Semi-)Automatically-generated Mappings

Innate to users is a need for information that is useful to them to complete some tasks. There exist at least two paradigms that describe information needs [Col11, Bro02]. The popular, widely-used computer science paradigm is that users seek to obtain answers that meet their needs by posing well-defined questions, in the form of queries, to an information system. Conversely, information science carries the view of information need as *'unknowable'* [Col11] and *'unspecifiable'* [Col11] by users, much like a *'black box'* [Col11]. In our research, we adopted the computer science paradigm because a need for information is often triggered from a requirement in the real world that is comprehensible by a human. This requirement, with the help of users, motivates the very use and construction of an information system as can be found in any system development models. The need for information is typically maintained by the user in the form of high-level human language and is translated into a form comprehensible by information systems to be retrieved, which is the query language. These suggest that users are equipped with the understanding of what information they require from an information system – supporting the computer science paradigm on information.

We direct the reader's attention to Section 1.1 for a working example of the challenge in identifying instances that can be relevant to a user's information need. Central to satisfying information need in a dataspace is the proper handling of semantic mappings. As described in Chapter 2, a semantic mapping represents a correspondence between elements in a global schema to a local schema, where

this correspondence indicates the notion of *similarity*. With semantic mappings being generated (semi-)automatically in dataspaces, the number of mappings can become numerous, with overlapping extensions, which may not satisfy the information need of a user.

We illustrate this condition with a naïve example. In this example we consider the task of selecting mappings as a black box. Let us consider three mappings that have been generated by a dataspace. The first, m_1, selects all records of European cities, while the second, m_2, contains information on African cities. Both mappings are of the type *basic mappings* [BPE+10]. After some time, a refinement takes place and m_3 is produced as a union of m_1 and m_2 (a *refined mapping* can be produced by combining existing mappings using one or more of the *union*, *join*, *difference*, *intersection* and *selection* operators [BPE+10]). As a result, cross-sourced mappings and single-sourced mappings with a narrower extent emerge. Later, a user poses a query, requesting information on European cities which have a population of more than 35,000. Through some selection process, m_1 and m_3 are selected for execution, and as a result, false positives are deposited into the result set through m_3.

Since not all mappings are relevant to a user's need of information, we view this problem as that of ranking mappings based on their relevance to a user's information need (RH1a). With ranking, *a*) a subset of mappings can be selected for evaluating a user's request [MPE12]; *b*) an order of evaluating mappings can be selected, where results are incrementally produced for user inspection [MPE12]; *c*) the number of mappings candidates can be reduced for other forms of pay-as-you-go manipulation of mappings, such as *refinement* and *selection*[1]; and *d*) the search space during instance integration can be reduced, which improves performance. Our hypotheses RH1b and RH1c (refer Section 1.3) state that semantic mappings can be ranked based on their relevance to a user's information need specified by the terms found in queries.

In this chapter, we present our proposed approach, which utilises a commonly-used relevance ranking technique, the TF-IDF which we have adapted to suit our needs. Additionally, we have conducted two empirical evaluations to test our approach, in order to understand *i*) how stable rankings can be achieved when different log sizes are used (RQ2 of Section 1.3); and *ii*) how the produced rankings track query patterns that are skewed towards specific sources (RQ3 in Section 1.3).

[1] *Mapping selection* is an entire research area of its own and is not part of our research. Recent work includes [BPE+10, HBP+11, FHH+09, EEL11, ABMM07, ACM+08, XE06].

This chapter is structured as follows. Section 3.1 introduces our proposal, followed by Section 3.2 that describes experiments designed for evaluating the feasibility of using implicit feedback to rank mappings. Section 3.3 illustrates existing work that are related to our approach. In Section 3.4 we briefly discuss the relationship of work in this chapter with our subsequent chapters. Section 3.5 summarises the whole chapter.

3.1 Our Proposed Ranking Approach

This research has the aim of investigating the use of pay-as-you-go approaches for instance level data integration, particularly that involve the identification of relevant instances and duplicates. The first of the two objectives that was formed to achieve this aim is to design an approach that ranks mappings based on their relevance to a user's information needs (refer O1 of Section 1.4).

Underpinning our approach is the hypothesis that a user's information need can be relayed by terms discovered in queries (RH1b), and in turn, these terms can be used to identify which mappings best fulfils this need (RH1c). Hence, this approach aims to filter out irrelevant records by ranking these mappings, so as to produce records that meet user's information need.

3.1.1 Using Terms from Query Logs to Rank Mappings

A user's request for information is assumed to be in the form of a query. A basic Structured Query Language (SQL) query can contain one or more *where* clause(s) which specify the *condition(s)* that a user wishes the system to adhere to. For example,

```
select name, address
from physicians
where position = "doctor"
and gender = "female"
and country = "United Kingdom"
```

where this query asks for the name and address of female medical officers residing in the UK. The use of the logical operator AND makes certain that if any of the conditions are not met, no records would be returned; while an OR logical operator relaxes this rule and allows for at least one of the conditions to be satisfied. The *equality* comparison operator suggests that the returned information

must satisfy a *precise* condition. Other comparison operators in SQL[2] include *inequality*, *greater than*, *lesser than*, *list of values*, *inclusive of two values*, *pattern matching*, *null test* and *non-null test*. For this proof of concept, we have chosen to investigate only on equality operator while the rest of the operators are left for future work. The reason being the equality operator specifically conveys the information that a user wants. In contrast, for example, an inequality operator per se conveys what a user does not require but without any helpful information about what exactly that the user needs. Hence, to use inequality necessitates further examination of the SQL SELECT statement which embeds it and demands a deeper understanding of their semantic association.

In principle, mappings are views that produce a set of records that satisfy a user's information need. In reality, this is not always true. A mapping can produce false positives in its result that may not satisfy this need of information. Nevertheless, we can relate terms found in queries to the results produced by mappings and hence can use these terms to rank mappings that produce the most relevant record set. We hypothesise that the rarity of terms found in a semantic mapping's extent can indicate how relevant the mapping is to a user's information need (RH1d). We use TF-IDF and a variant to give scores to the mappings in order to rank them.

3.1.2 Term Frequency/Inverse Document Frequency

TF-IDF is a weighting scheme that is widely used to rank documents based on their relevance to some search terms. A term t in document d can be assigned different weights, depending on the following rules.

- Lowest weight when t appears in nearly all documents, implying that it has no discriminating effect;
- Lower weight when t scarcely exists in a document or exists in many documents; and
- Highest weight when t is frequently present in a small number of documents, which indicates the differentiating role of t.

TF-IDF builds upon two weighting schemes, the Term Frequency (Equation 3.1) and the Inverse Document Frequency (Equation 3.2), where Term Frequency is the number of occurrences of term t in document d,

[2]http://www.tutorialspoint.com/sql/sql-operators.htm

$$tf(t,d) \tag{3.1}$$

and Inverse Document Frequency is defined as,

$$idf(t) = log\frac{|D|}{df(t)} \tag{3.2}$$

where $|D|$ is the total number of documents in the corpus D, and $df(t)$ is the number of documents in D that contain t (a.k.a. document frequency). Here, *idf* is used instead of regular document frequency. This is to further reveal rare terms and award them with a high weight, and simultaneously to soften any influence that frequent terms have, hence, a low weight is given.

In Table 3.1, we echo the example presented by Manning [MRS08]. The example compares the effects of document frequency (*df*) and inverse document frequency (*idf*) when seeking relevance from a collection of documents owned by Reuters, which has a total of 806,791 documents. With *df*, the rare term *auto* receives the lowest value because it does not frequently appear in many documents, while *best* was counted to occur 25,235 times; but with *idf*, *best* is not viewed to be discriminative and does not help in narrowing down the search to documents that are more likely to focus on it as the main subject.

Table 3.1: Comparison of *df* and *idf* values of Reuters collection [MRS08]

Term	***df***	***idf***
car	18,165	1.65
auto	6,723	2.08
insurance	19,241	1.62
best	25,235	1.5

By combining term frequency and inverse document frequency, TF-IDF (Equation 3.3) can be defined as,

$$TF\text{–}IDF(t,d) = tf(t,d) \times idf(t) \tag{3.3}$$

which specifies that TF-IDF is the product of the Term Frequency and the Inverse Document Frequency. This score increases in proportion to the number of times a word is found in the document, but is offset by the inverse frequency of the word in the corpus, which adjusts to *stop words* (frequently-appearing words e.g., *the, is, that*, etc.).

3.1.3 Applying TF-IDF to Score Mappings

To apply TF-IDF in our setting, we ignore any structure in the mapping's result set, and represent it as an unstructured document. Terms found in query logs which are used with an equality operator are extracted and placed as a search string. We adopt the view that terms in queries are evidential signs of a user's interest in seeking them regardless of the setting. As an example, assuming a user posts a query for *S. pombe* genes over a *gene* table, we should expect that the user may also be interested to find *S. pombe* proteins in proteomics experiments, where such an interest is expressed in a different query. Also, we disregard any *stop words*, considering the values were originally collected for database attributes where mostly were short and concise descriptions as compared to longer and fuller sentences of the natural language. We show in equation 3.4 how we apply TF-IDF to produce a score for each mapping. A mapping belongs to a collection of mappings that a dataspace has, regardless by manual or (semi-)automatic method of generation. Scoring mappings using TF-IDF involves scanning through the extent of each mapping m (i.e., $extent(m)$) and calculating the frequency of each relevant term t found in query log l. An extent of a mapping consists of a set of instances that corresponds to the criteria defined by the mapping's underlying query definition.

$$TF\text{–}IDFScore(l, m) = \sum_{t \in l} TF\text{–}IDF(t, extent(m)) \tag{3.4}$$

Size-normalised TF-IDF for Scoring Mappings

An important notion relating to a mapping is its *size*. The size of a mapping is the total number of terms that exists in a mapping's extent to describe the mapping's condition of being *larger* or *smaller*. The portion of relevant terms found in a mapping's extent is relative to the mapping's size (Equation 3.5).

$$Portion\ of\ relevant\ terms = \frac{Total\ relevant\ terms}{Mapping\ size} \tag{3.5}$$

It is important to take into account that a larger mapping may not necessarily contain a higher portion of relevant terms when compared to a smaller mapping. We illustrate this condition using the following case.

Case 1: Consider two mappings *m1* and *m2*, where *m1* has a size of 1000, indicating that it contains 1000 terms. From *m1*'s size 100 are relevant terms. *m2*

has 50 relevant terms out of a total of 200 terms. If all terms are assumed to be equally discriminating, and if both mappings have the same *idf* value, e.g. 0.1, by using this formula the larger mapping, *m1*, would be ranked higher. Using the TF-IDF formula in equation 3.4, we calculate the ranking score for mappings *m1* and *m2*.

$$Score_{m1}: \quad 100 \times 0.1 = 10$$

$$Score_{m2}: \quad 50 \times 0.1 = 5$$

Viewing at a finer level, we could see that *m1* has only 10% relevant terms while *m2* has 25%, thus ranking *m1* higher than *m2* is counterintuitive.

$$Relevance_{m1}: \quad \tfrac{100}{1000} \times 100\% = 10\%$$

$$Relevance_{m2}: \quad \tfrac{50}{200} \times 100\% = 25\%$$

To capture this fraction of relevance of a mapping's extent, we now propose a variation to TF-IDF, which aims to normalise the effects of a mapping's *size* on the decision to rank. We define the *Size-Normalised* (SN) TF-IDF (Equation 3.6) as:

$$TF\text{–}IDFScoreSN(l,m) = \sum_{t \in l} \left(\frac{TF\text{–}IDF(t, extent(m))}{size(m)} \times log(size(m)) \right) \tag{3.6}$$

where *size(m)* is the total number of terms, relevant or otherwise, which are contained in a mapping's result. The result that we would obtain would be in favour of a mapping that has more relevant terms, which in our example is *m2*, instead of mappings that contain more terms. The introduction of *log(size(m))* in the equation is to magnify the value of the scores to better see the subtle characteristics of the graph. An application of using logarithm with the same purpose was found in [KHWL93]. The difference is the values were expressed as a logarithm. In our scenario, this approach was found unsuitable because it yielded negative values. As a result, we use logarithm as a multiplication to the scores.

$$Score_{m1}: \quad \frac{(100\times0.1)}{1000} \times log(1000) = 0.03$$

$$Score_{m2}: \quad \frac{(50\times0.1)}{200} \times log(200) = 0.057$$

We do not claim that one scoring scheme is better than the other, but instead the distinct features of the resulting mappings are highlighted here, leading to the use of both scoring schemes in our experiments.

3.2 Experiments

We present an empirical evaluation that we have conducted on our proposed approach with the aim of answering the questions of *a*) how quickly stable rankings can be achieved when different log sizes are used (RQ2); and *b*) how the produced rankings track query patterns that are skewed towards specific sources (RQ3).

3.2.1 Experimental Setup

Data from two publicly-available life science databases were used for this investigation (the Stanford Microarray Database and ArrayExpress), which describes experiments conducted on *Saccharomyces Cerevisiae* (a.k.a. bakers yeast). These data were uploaded into a Postgres DBMS, where one table represents the experiments in ArrayExpress and two tables represent the *Experiment* and *Result* tables of the Stanford Microarray database. Additionally, another table was created to represent a global schema.

Experimentation was conducted on ten mappings which we have manually created, where three are *basic mappings* (directly map the source tables to the global schema), and there are seven *refined mappings* (combine mappings using a *union*, *join*, *intersection* or *selection* operator [BPE+10]). Resonating with existing mapping-generating tools, e.g., Clio [FHH+09, HHH+05], our mappings were formed to exhibit a systematic exploration of alternative derivations of mappings, where refined mappings are built from the basic mappings with the use of a collection of operators. In this spirit, we have applied the *difference* operator in Mappings 7 and 8 (Table 3.2) where for each mapping the first set is Mapping 4 and 5 respectively, and the second set is a collection of nameless genes stored in an auxiliary table. These genes, to which biologists have not given any names, go by their Open Reading Frame (ORF) identifier, e.g., *YHR089C*. Another manner of producing refined mappings is by combining existing refined mappings using

Table 3.2: Mapping descriptions

Type	Mapping	Description
Basic	1	SELECT all experiments from Stanford Experiment table.
Basic	2	SELECT all experiments from Stanford Result table.
Basic	3	SELECT all experiments from ArrayExpress table.
Refined	4	JOIN tables Experiment and Result in Stanford and SELECT experiments of type limit.
Refined	5	SELECT experiments of types limit, growth, genotype and time in ArrayExpress.
Refined	6	UNION result sets from Stanford (Mapping 4) and ArrayExpress (Mapping 5).
Refined	7	SELECT experiments in Stanford (Mapping 4) and perform a DIFFERENCE with a relation containing nameless genes.
Refined	8	SELECT experiments in ArrayExpress (Mapping 5) and DIFFERENCE with a relation of nameless genes.
Refined	9	UNION Mapping 7 (Stanford) and Mapping 8 (ArrayExpress).
Refined	10	SELECT experiments conducted on yeast gene located on the Watson strand from Mapping 9.

Table 3.3: Mapping sizes based on total number of terms

Mapping	Size
1	18,047
2	36,561
3	106,876
4	199,737
5	75,379
6	255,940
7	182,213
8	73,031
9	255,201
10	126,052

the earlier stated operators [BPE+10]. We performed this by producing Mapping 6 as the *union* of Mappings 4 and 5, and the *union* of Mappings 7 and 8 to produce Mapping 9.

Table 3.3 shows the sizes of all ten mappings based on the number of terms in each mapping's set of results. The sizes are in accord to the operations conducted to form the mappings. For example, Mapping 5's size is 75,379 as it is a subset of Mapping 3 which has a size of 106,876. Another example is Mapping 6, having size 255,940, is the result of a union between Mappings 4 (size 199,737) and 5 (size 75,379) with duplicates removed. Our assumption is a single data source does not host duplicates but they exist across data sources. The decision to not consider duplicates is because by having them the rarity of some terms would dampen, thus influencing the resulting order of rankings. These characteristics

Algorithm 1 Query log file generator for Experiment 1

```
1: Create log file log if it does not exist
2: for i = 1 → 100 do
3:     columnname = select random column from global schema
4:     tuplelist = get all tuples from global schema where column name equals to columnname
5:     value = select random tuple from tuplelist
6:     append value into log
7: end for
```

are also inherent in Mapping 9.

Determining relevance of mappings is a difficult task. Each user has a specific information need in the form of a ground truth that cannot be generalised over all users. Based on this premise, we evaluate not the *correctness* of the ranking (i.e., the determination of which mapping is more relevant than another), but instead focus on evaluations that inspect *a*) the *stability* of the produced rankings when feedback amount gradually increases (RQ2); and *b*) the *extent* to which the rankings are able *to reflect* the concentration of query patterns in a particular source (RQ3).

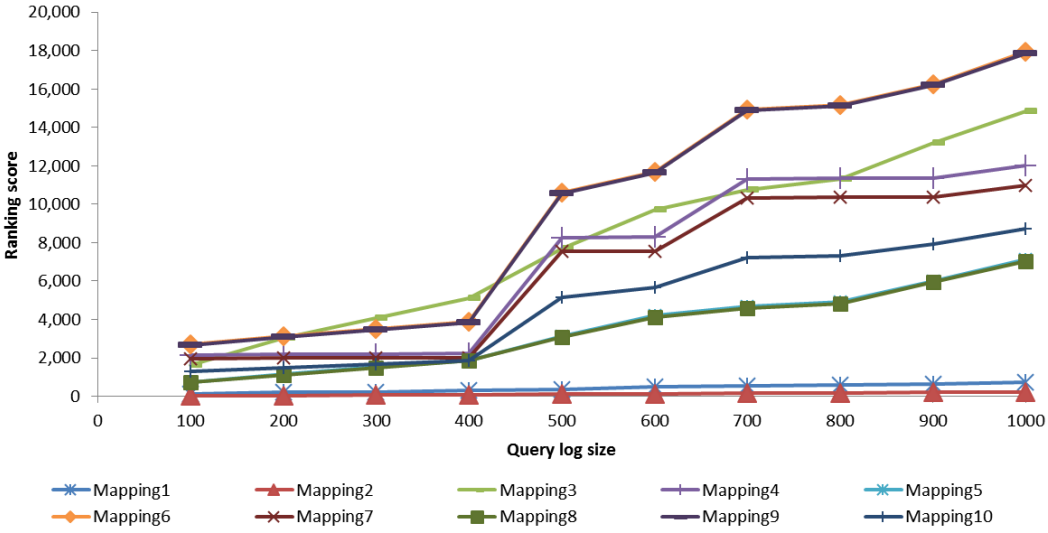
3.2.2 Experiment 1: Effects of Varying Query Log Size

Our first evaluation investigates the question of *how different sizes of query log affect the ranking score* (RQ2). In other words, what is the amount of implicit feedback that would produce stable ranking? We describe the expression *stable* as a situation where the order of mappings, in rank, is generally consistent across varying sizes of query logs.

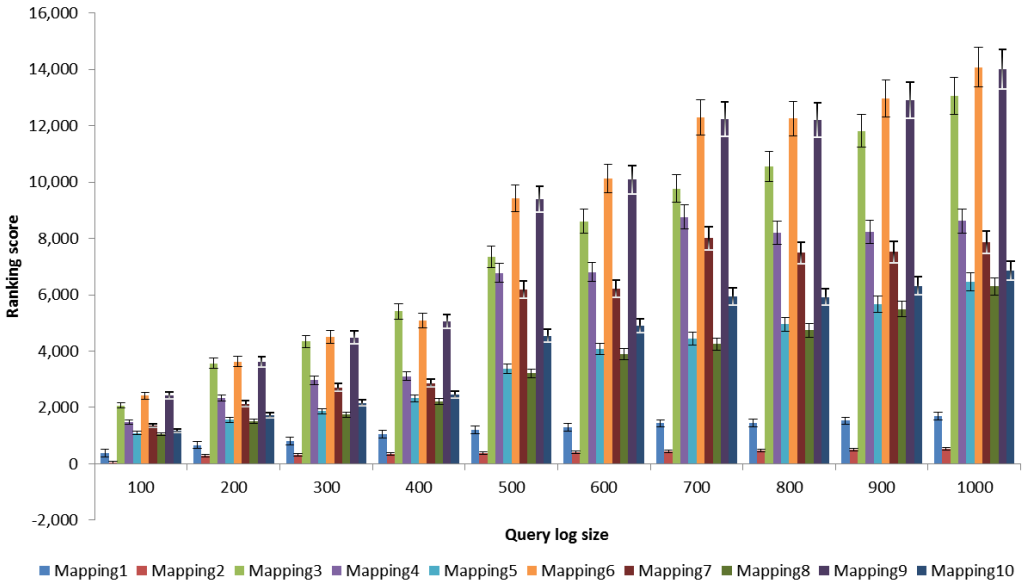
To perform our evaluation, we generated synthetic queries from our homegrown generator. The random terms injected into each of these query are extracted from a global schema and each term simulates a literal that can be found in a query's conditional clause. As we have stated earlier, in our research, we only consider equality-based conditional clauses hence the synthetic queries we have generated are exclusively of this type. A more detailed description of this extraction process is provided in Algorithm 1. Emulating the growth of a log file over different points in time, we produced 10 log files to represent the different snapshots. The smallest file has 100 values, and for each consecutive file an increment of 100 was applied, until the largest file of 1000 values was produced.

We calculated the ranking by using both of the described scoring schemes; each evaluation was run five times and the average of the scores was plotted in Figures 3.1a and 3.1b. To see how precise our estimates are of the scores, we

calculated the variance and present it in Figures 3.2a and 3.2b. The following are descriptions of the results.



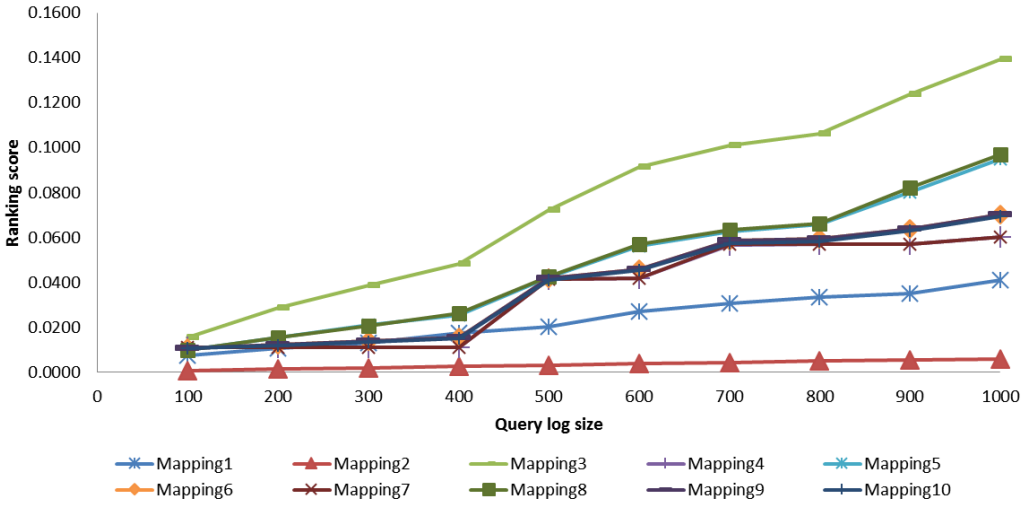
(a) Rankings



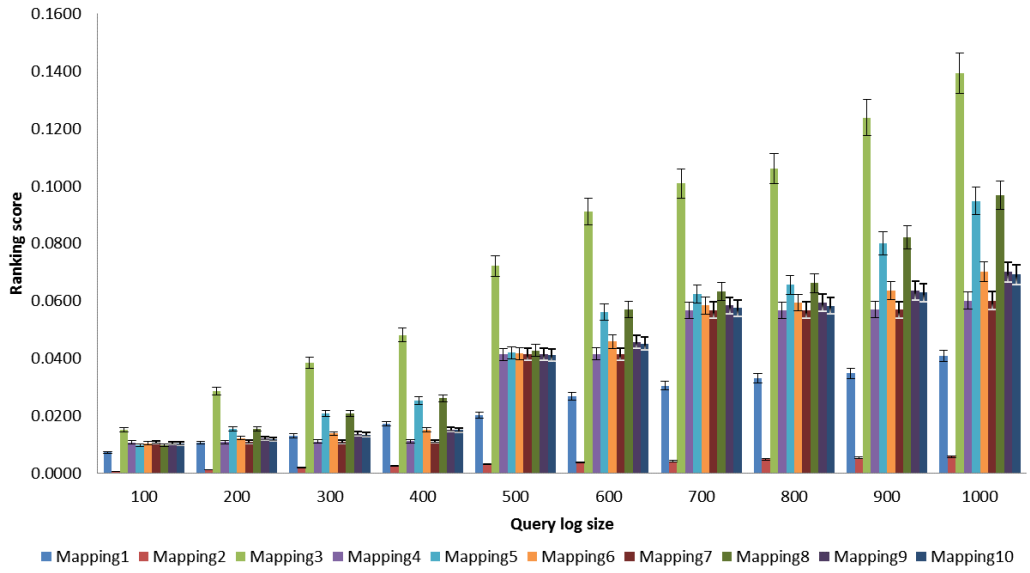
(b) Variance

Figure 3.1: TF-IDF Ranking Score for All Mappings.

- From Figure 3.1a we can observe that there is an upward trend in the TF-IDF scores for all mappings as more terms are introduced through the individual query logs. This indicates that a TF-IDF's score hinges upon the total of the scores that each term in the log obtains.



(a) Rankings



(b) Variance

Figure 3.2: Size-Normalised TF-IDF Ranking Score for All Mappings.

- We could also observe that most of the mappings generally retain their stability across different log sizes and stability was achieved even from quite small query logs, as evident from log size 500 and larger. Stability describes the condition where the order of the rankings is generally unchanged.
- There is a noticeable surge of TF-IDF score for mappings 6, 9, 4, 7 and 10 of Figure 3.1a, particularly between log sizes of 400 to 500. Closer inspection revealed that the mappings uniformly contain a few terms that incidentally appear in great numbers in the extent of the mappings, contributing to the high term frequency score; while many of the terms in the extent have low frequency.
- The varied TF-IDF scores of the mappings, where some are much higher than another, highlights the dependency of term frequency on mapping size.
- Mappings that received top rankings are not influenced by the sources that they described but by the randomly selected terms that they contained and the number of these terms in their extents.
- From Figure 3.1b, we can observe that in general the scores of all mappings have a small variance, suggesting a high confidence of the preciseness of our estimates of the scores.

Figures 3.2a and 3.2b show the results of our evaluation when mapping size is normalised.

- From Figure 3.2a we recorded a rise in the size-normalised TF-IDF score for all mappings when increasing sizes of query logs are supplied. With size-normalised TF-IDF, Mapping 3 is considered to be more relevant than Mapping 6. Owning fewer relevant terms than Mapping 6, Mapping 3 however, has a higher fraction of relevant terms in relation to its mapping size of 106,876 while Mapping 6 has a size of 255,940.
- Stability-wise, from quite small-sized query logs, this was achieved with size-normalised TF-IDF. This is demonstrated by the steady upward trend in the score, and in this case stability is reached slightly earlier at log size 400.
- The presence of sharp increases in the score is because of the high frequency that some terms produce. This is also found in the non-normalised TF-IDF.
- Top ranking mappings are affected by the fraction of relevant terms found in their extent, whereby these terms were chosen at random during query

generation. Hence, there is no correlation between a mapping's ranking with its underlying data source.

- In terms of how precise our estimates are of the produced scores, from Figure 3.2b we can observe that a small variance is found in all mappings. This suggests a high confidence that our estimates of the scores are precise.

From the results obtained in Experiment 1, we can make several conclusions. The first conclusion is that both scoring schemes produced stable rankings at small query log sizes. The second conclusion is each scoring scheme portrayed different features of the produced mappings, thus we make no claim that one scheme is better than the other. The third conclusion is small variance was found in our generated ranking scores, suggesting a high confidence in the preciseness of our estimates from the two scoring schemes.

3.2.3 Experiment 2: Effects of Varying Query Log Skew

To fulfil the second part of our objective, we investigate *how rankings track query patterns that are skewed towards specific sources* (RQ3). The premise is that there are patterns more used in certain data sources than others, which forms a concentration of terms. The question is whether the proposed ranking technique would be able to reflect this skewed characteristic.

We define the skew of a query log l in relation to a source s to be the fraction of the queries in l that feature terms that are exclusive to s (Equation 3.7):

$$Skew(l, s) = \frac{N_{ls}}{N_l} \tag{3.7}$$

where N_{ls} is the number of queries in the query log that reference only terms in s and N_l is the total number of queries in the log.

To evaluate the ranking behaviour when dealing with skew, 21 query logs were produced from the log file generator described in Algorithm 2. Each file holds 500 queries which contain a different number of embedded terms that are exclusive to one of the data sources, demonstrating its level of *skew*. For example, a log containing 50 terms exclusive to ArrayExpress suggests an *ArrayExpress-skew of 10%*, complemented with 450 Stanford-exclusive terms which depicts a *Stanford-skew of 90%*. We casted an interval of 5% skew across the generated 21 log files, such that the first file presents 100% Stanford-exclusive queries, followed by a log file of 5% ArrayExpress-exclusive queries and 95% Stanford-exclusive queries, ending with a log file that contains 100% ArrayExpress-exclusive queries. No

Table 3.4: Mappings and their exclusivity to one or more data sources.

Mapping	**Exclusivity**
1	Stanford
2	Stanford
3	ArrayExpress
4	Stanford
5	ArrayExpress
6	Neutral
7	Stanford
8	ArrayExpress
9	Neutral
10	Neutral

non-exclusive queries were selected. The reason behind this is the adding of non-exclusive queries would make the controlling of skew level to become difficult.

Exclusivity to a particular data source applies not only to queries in logs but also mappings. In Table 3.4 we list down all ten mappings and their exclusivity to which data source. Mappings 6, 9 and 10 have a *neutral* exclusivity due to the fact that their underlying data sources are both Stanford and ArrayExpress.

Algorithm 2 Log File Generator for Skewed Logs

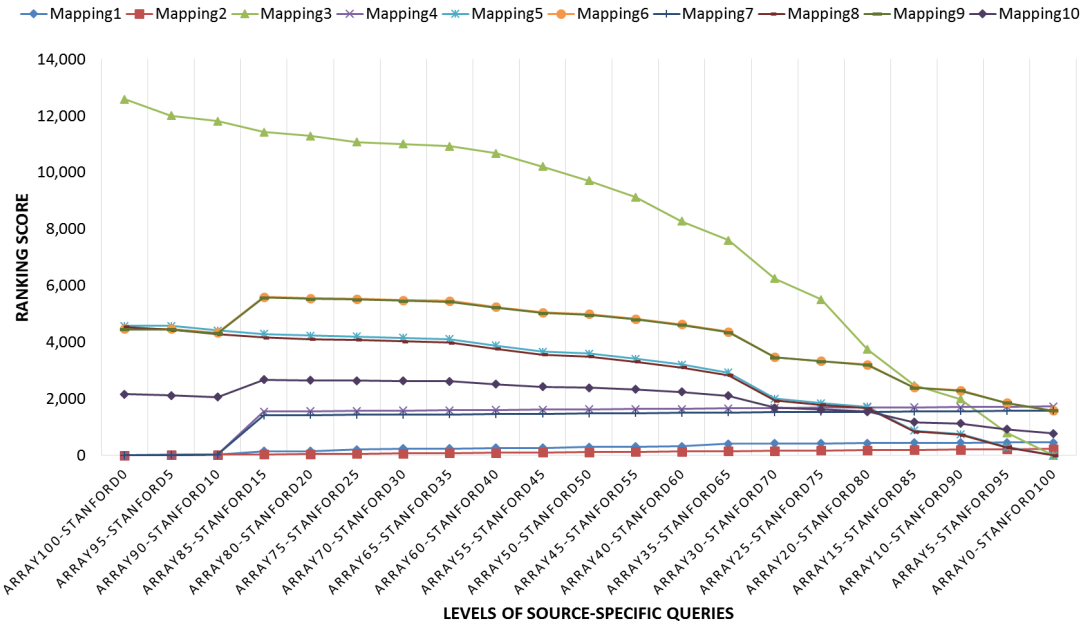
1: **Input:** *skew1* integer
2: **Input:** *skew2* integer
3: Create log file *log*
4: **for** $i = 1 \rightarrow skew1$ **do**
5: *valuelist* = get all values from the global schema produced by (Mapping 1 or Mapping 2) but not Mapping 3
6: *value* = choose a value at random from *valuelist*
7: append *value* into *log*
8: **end for**
9: **for** $i = 1 \rightarrow skew2$ **do**
10: *valuelist* = get all values from the global schema produced by Mapping 3 but not (Mapping 1 or Mapping 2)
11: *value* = choose a value at random from *valuelist*
12: append *value* into *log*
13: **end for**

Five runs were made and the average was calculated. Figures 3.3a and 3.4a present the trend of the resulting rankings when each of our scoring schemes is used, while in Tables 3.5 and 3.6 show the rankings' order and their exclusivity to a specific data source. We display the variance in Figures 3.3b and 3.4b. For better viewing, we included in Appendix A Figure 3.3a divided into four quarters across the different levels of the query logs and in Appendix B we displayed Figure 3.3a in a similar manner. The following is the full description of our obtained result.

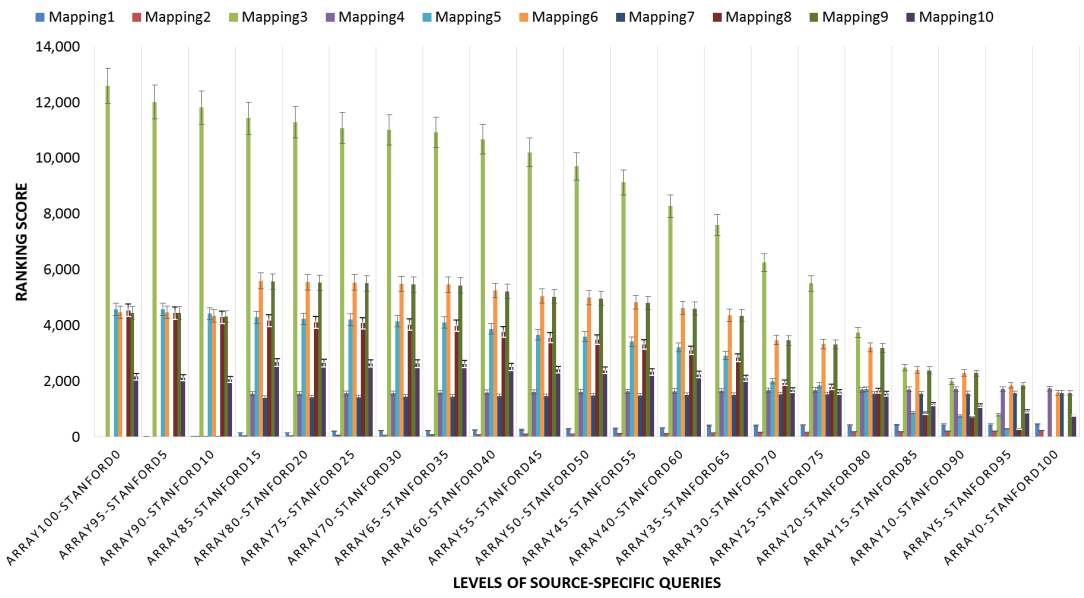
- We observe that the TF-IDF score for basic mappings, Mappings 1, 2 and 3, grows steadily as the skew towards its specific source increases. This could also be observed with refined mappings for their specific skew source.
- Neutral mappings are fairly stable in their order of rank as the query logs become more skewed (Tables 3.5 and 3.6).
- Comparing the rankings at 50%-50% skew with our first experiment, we can see that the TF-IDF scores are significantly lower. This is caused by the exclusivity of terms in the query log to only one data source, reducing the number of times a term can be found. In other words, terms shared between both sources, which can be found in Experiment 1, are no longer present, leaving a smaller group of terms, with lower average frequencies.
- In both the results for TF-IDF and size-normalised TF-IDF, Mapping 3, which is biased to ArrayExpress, was ranked highest even when the skew is towards Stanford. This can be seen with log file having ArrayExpress-prone terms as small as 20% when TF-IDF scoring is used; and 15% for size-normalised TF-IDF scoring. The cause in the former scoring strategy is the presence of the largest amount of relevant data, while taking into account that some terms are more discriminating than others; and in the latter scoring strategy the demonstration of the highest fraction of relevant data, also taking into account that some terms are more discriminating than others.

  Such skew is due to properties that are unique to each mapping in relation to the range of values that the mapping contains. Other score-determining factors include a mapping's size and frequency of terms.
- From Figures 3.3b and 3.4b, it could be observed that we obtained a high confidence that our estimated scores are precise from the small variance found in all ten mappings.

Several conclusions can be drawn from the results obtained through Experiment 2. Firstly, our TF-IDF ranking scheme and Size-Normalised TF-IDF ranking scheme were able to reflect the skewness of the query logs. Secondly, because of similar reasons to Experiment 1, we again do not claim that any of the scheme is better than another in reflecting skewness. Finally, high confidence was acquired as to how precise our generated estimates of ranking scores were under both schemes.

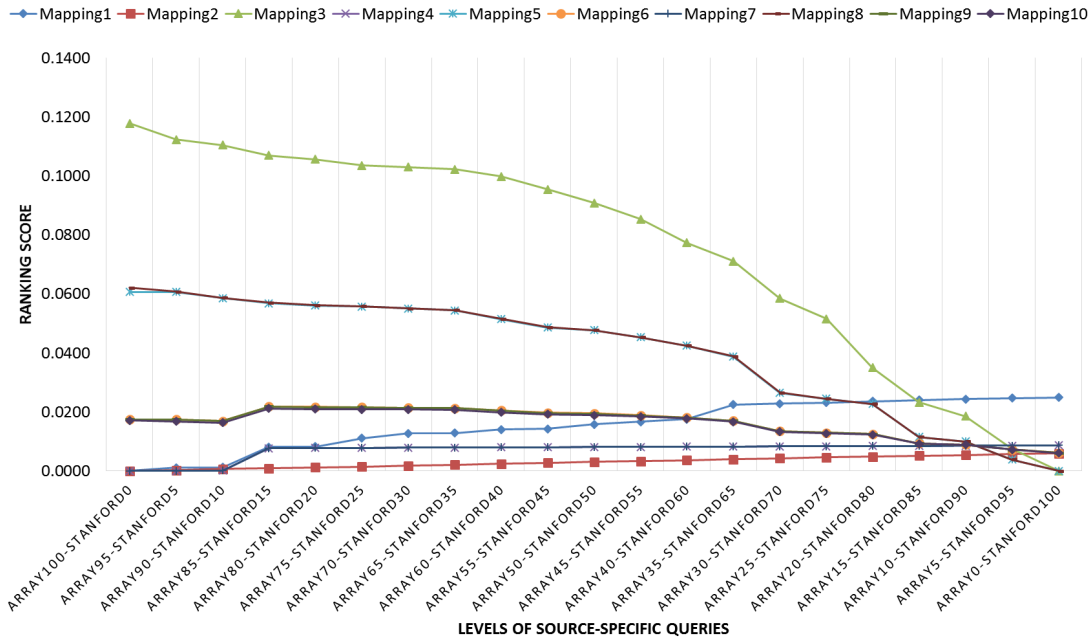


(a) Rankings

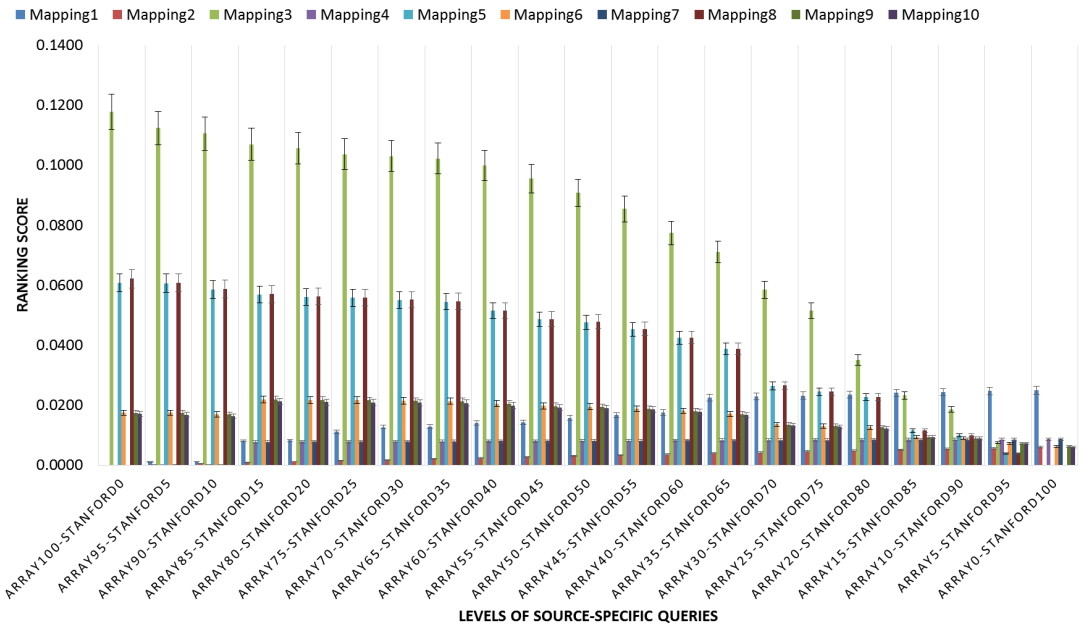


(b) Variance

Figure 3.3: TF-IDF scores for different levels of skew.



(a) Rankings



(b) Variance

Figure 3.4: Size-Normalised TF-IDF scores for different levels of skew.

Table 3.5: Ranks Obtained using TF-IDF Scores.

Mapping	**Exclusivity**	**100% Array Express**	**50% Stanford 50% Array Express**	**100% Stanford**
1	Stanford	7	9	6
2	Stanford	7	10	7
3	ArrayExpress	1	1	8
4	Stanford	7	7	1
5	ArrayExpress	2	4	8
6	Neutral	4	2	2
7	Stanford	7	8	3
8	ArrayExpress	3	5	8
9	Neutral	5	3	3
10	Neutral	6	6	5

Table 3.6: Ranks obtained using Normalised TF-IDF ScoresAll Mappings.

Mapping	**Exclusivity**	**100% Array Express**	**50% Stanford 50% Array Express**	**100% Stanford**
1	Stanford	7	7	1
2	Stanford	7	10	7
3	ArrayExpress	1	1	8
4	Stanford	7	8	2
5	ArrayExpress	3	2	8
6	Neutral	4	4	5
7	Stanford	7	9	3
8	ArrayExpress	2	3	8
9	Neutral	5	5	5
10	Neutral	6	6	6

3.3 Related Work

In this section, we discuss existing work related to our proposal, with the goal of exhibiting our contribution in filling up a gap in knowledge, specifically, the ranking of semantic mapping in a pay-as-you-go data integration setting. For our discussion on implicit feedback, readers are referred to Section 2.5.2.

3.3.1 Ranking

The term *rank* or *ranking* is frequently being related to search engines and web pages. For example[3], when a search is carried out on "microphones", a website's

[3]http://www.webopedia.com/TERM/R/rank.html

ranking specifies the exact *position* it is in the search results: e.g., within the top 5, on the first page, the 300th page and so on. The sort of ranking that we do in our research follows the spirit of search engine ranking but by a different method, application and environment. An important notion is the correctness of the produced ranking. In many cases, ranking can merely suggest possible relevance, because only a user can determine the correctness.

Driven by the active needs of the database and information retrieval communities, ranking employs different strategies. Although the expected consequence in both areas is a set of prioritised answers in light of a user's request, applying a single solution to both areas is difficult. In databases, structure plays a significant role in the ascertaining of a term's relevance to a user's query. If a user searched for a term *python* in an attribute *animal*, it is irrelevant to another user who wishes to find *books* on the Python programming language. In contrast, structure is absent in information retrieval with the entire document needed to determine a term's relevance. Therefore, a term's location in the document is significant although it can be difficult to exploit.

The type of query conducted for structured data directly influences the form of ranking that is available or that may be appropriate. To conduct *keyword queries* over a structured database, current proposals [ACD02, BHN+02, HGP03, HP02] typically engage a graph-based model to represent the underlying structured database. Once terms have been extracted from a user's query, the graph is explored to seek relevant records. In most cases, posted queries necessitate the joining of multiple tables, encouraging the formation of paths that join tuples from more than one table, simply known as a *tuple tree* or *join tree*. Ranking of records must be preceded by the ranking of the tuple trees, commonly involving the calculation of joins forming the tree. While copious joins may intimate the presence of less relevant results, a small number of joins tends to indicate higher relevance since they are more likely to contain pertinent results [HP02]. Although we use query logs as unstructured documents that are accessed by keyword-based queries, our work taps into the underlying structured database through a layer of abstraction.

Recurring problems in structured queries, but not exclusive to them, are the *Many-Answer problem* and the *No-Answer problem*. The former happens when the conditions of a query are too general, hence, an abundance of records is returned. Meanwhile, the latter occurs when a query has very strict conditions, and thus does not produce any result. Of the two, in our view, the Many-Answer

problem closely resembles our multitude of mapping results, in terms of the number of the produced records to be ranked. In general, ranking in structured queries centres around solving either of these problems. Current proposals borrow different techniques from the information retrieval domain in their attempts to solve the problems: *a*) adapting TF-IDF to harness its discriminative ability [AEKV07]; *b*) adapting and applying probabilistic models to manipulate *intrinsic* information of the database's underlying structure e.g., the correlation between attributes that were specified in a query and ones which were not (i.e., unspecified attributes) [CDHW06] and calculating a *global score* which represents the *global importance* of an unspecified attribute [CDHW06].

We now review existing proposals engaged in ranking using some form of implicit user feedback.

Page ranking algorithms [DSB09] place millions of websites into an ordered sequence based on their importance and/or relevance to a user's search query. In parallel to the increasing number of users, the number of queries have also grown exponentially. Although in our case we assume the manipulation of data by a single user, however, we expect many queries to be posted throughout the lifetime of a dataspace, hence, it is useful to turn to *page ranking algorithms* for comparison. There are a number of different proposals for ranking of web pages, as presented in a survey by Duhan *et al.* [DSB09]. Some of these techniques mine the contents of websites (i.e., Web Content Mining) to determine ranking, while others discover hyperlinks (i.e., Web Structure Mining) or user navigation pattern (i.e., Web Usage Mining). A similarity to mapping result sets is that websites contain a collection of terms that are potentially relevant to users, however, a dissimilarity is the presence of hyperlinks in web pages which traditionally function as a means for navigation. Duhan *et al.* differentiate between the *importance* of a website and its *relevance*. Importance is described as the popularity of the website where a larger number of incoming links indicates greater popularity; relevance is pegged to the prominence of terms found in a page with respect to a given query. From among the algorithms in Duhan *et al.*, only Page Content Rank (PCR) [PS05] fully regards relevance as the primary characteristic for ranking, therefore, we focus our review on it.

Fundamental to PCR is a set of heuristics regarded essential in the analysis of websites. Although we have stated earlier that PCR is the only surveyed proposal which fully considers relevance, elemental to this measure of relevance

is the notion of importance. The more important a page is, the more relevant it is to a user's query, and hence, the higher is its ranking. Several dimensions of importance have been adopted into the final relevance score, which we will describe as we go along.

PCR is calculated through the execution of four processes. At the start, terms are extracted from every page in D, where D is the set of all pages indexed by a search engine. An inverted list of every term is built here for use in the final process. In the second process, statistical parameters are computed. These parameters are term frequency (i.e., the number of times a term t appears across a set of documents D. We view this definition to be close to collection frequency rather than the earlier introduced term frequency within the context of TF-IDF), incidence of pages (i.e., the ratio of the number of documents that contains t over the total number of documents), frequency of words in the natural language, synonym classes, the distance between t and the terms in a query Q (occurrence positions), and the prominence of t's neighbours which influences t's prominence i.e., neighbour's prominence. The next process aims at determining the importance of every term found at the start. A neural network is employed as the classifier: each parameter is associated with one input neuron, and a term's importance is specified with a single output neuron. Calculating a relevance score is the goal of the final process. The multiple dimensions of importance are consolidated here through the forming of a single relevance score. Thus, we can equate this new score of page P as the average importance of terms found in P.

Now we discuss the set of statistical parameters used in PCR, which either carries the different dimensions of importance or contributes to its novelty. First is *the distances of occurrences of t from occurrences of terms in Q (set of occurrence positions)*. The intuition behind the use of this parameter is if t frequently occurs or is frequently located close to terms in Q, then t is significant to the search executed through the use of Q. Hence, the measurement that this parameter demonstrates is *the distance of t from a set of query terms posted, Q, can be quantified by the minimum distance computed from the entire existing distances of t with every query term in Q*. The second parameter is *incidence of pages*, where it seeks to know what is the fraction of pages with t (a.k.a. document frequency) to the total number of pages. This indicates that PCR regards terms that have high document frequency to be more important than a term that has high term frequency. The next parameter measures the frequency of a term in the natural language. PCR assumes that there is an available database of frequently

used words in natural language. If t is found in the database, hence, it is assigned a lesser importance value for it is more likely to be a *stop word* e.g., *the, this, is*, etc.

Another interesting parameter used in PCR is *synonym classes*. The idea is that there exists a database that contains classes of synonymous words. Each class has its own meaning; a single meaning is shared by members of the same class. It is only logical that if t is a synonym to an important term s, then t should also be regarded as being as important as s, as well as other synonymous terms found in the class. Hence, the importance of t is the aggregated importance of all terms in the synonym class it belongs to. The last parameter essential to the usefulness of PCR is the *importance of neighbouring term*. A term t that is constantly surrounded by important neighbours acquires importance through the aggregation of its neighbours' importance value.

Since PCR targets Internet pages that contain lengthy, connected text of description that is close to natural language, a hefty amount of processing was spent on manipulating these texts. In our case, text is shorter and forms small, discrete units of description after we removed the underlying database structure and regarded each mapping's result set as an unstructured document.

Probabilistic information retrieval approach (PIRA) [CDHW06] aims at solving the *Many-Answer problem*. Considering the combined number of records that multiple mappings can produce, we presume its amount to be of the scale associated with the Many-Answer problem, which makes PIRA potentially relevant. PIRA operates by the intuition that ranking functions must take advantage of information that can be gained from attributes that were not specified in a user query (i.e., unspecified attributes), due to the fact that since in the Many-Answer problem all user-required conditions are fully met, the discriminative property that a ranking function is expected to support is absent. Such information conveys not only the *correlation* between a *'specified'* attribute and its *'unspecified'* counterpart but also how important an *'unspecified'* attribute's value is at a *global* level. We illustrate both types of information using an example taken from Chaudhuri *et al.* preceded by a scenario that describes a case of *correlated attributes*, and followed by a scenario on how an attribute value is perceived as *globally important*. Suppose a database of homes for sale has the following structure.

```
(TID, Price, City, Bedrooms, Bathrooms, LivingArea, SchoolDistrict,
```

`View, Pool, Garage, BoatDock ... )`

Consider a query which requires the following conditions.

`City = 'Seattle' and View = 'Waterfront'`

A correlation to an unspecified attribute `BoatDock` suggests that buyers who ask for a waterfront view would most likely also be interested to own a boat dock. Therefore, records with `BoatDock='Yes'` would receive higher ranking than one where `BoatDock = 'No'`.

We continue with our second example. Consider a home located in a place which receives,

`SchoolDistrict = 'Excellent'`

This would cause that home to be highly ranked because it is globally desirable to have a home which has school districts that are good.

Like us, each record in PIRA is treated as a *"document"*, suggesting that relevance is applied on the records. To quantify the relevance of a record in light of a user's query, PIRA suggested the automatic generation of scores for each record based on query workload and data analysis. A probabilistic method commonly used in information retrieval, specifically Bayes' Theorem, was used to calculate the probability that a particular correlation is relevant depending on evidence found in the workload. The record with the most probable relevance is ranked highest.

Three main processes support the probability-determination task. In the first process, any correlations between specified and unspecified attributes are discovered. Such discovery is performed by following a simple but constrained assumption (i.e., limited conditional independence) that says, *given a query Q and a tuple t, the specified attribute values, X, within t are assumed to be independent, though dependencies between X and their unspecified attribute values, Y, are allowed.*

Any *functional dependencies* that occur within X or Y should be brought to attention. This concern is undertaken by the second process. Functional dependency is an association between an attribute a_i and another attribute in the same table, a_j such that which a_j has the ability to determine the uniqueness of a_i. For example, "$Zipcode \rightarrow City$". Also here, any existence of transitive closure related to the functional dependencies gets computed. Such information

is obtained by investigation of records in a database, and in some unspecified cases these correlations are *"tuned"* by domain experts.

This final and central process addresses the numerical estimation of relevant records sought from workloads. Often, this estimation is received from user feedback at query time or from some training activities, but Chaudhuri *et al.* propose the use of the workload for identifying relevant records. PIRA taps into existing information retrieval models aimed at leveraging query-log information, such as [RJ05, STZ05], to complete this estimation exercise. The product of this process is a two-part score, where one part measures the global importance of unspecified values e.g., waterfront, greenbelt and street views, whereas the other measures the correlation or dependencies between these values and the specified attribute values, e.g., *City = "Kirkland"* and *Price = "High"*. Finally, records are ranked based on the score they received.

PIRA's strength in ranking records is in the availability to use reliable mass of past information traced from queries. This supplies information in aggregated form (i.e., query counts), necessitating a great deal of compute time and effort. Access to logs of queries is also conducted in our work; what differs from PIRA is that we extract single occurrences of each term found in the log file to be used in deciding the ranking order leading to reduced compute time. Although PIRA identifies functional dependencies automatically, to some degree, domain experts are still sought to *"tune"* the correlation further. This reliance on functional dependencies as part of its central processes implies that a great amount of compute time is needed, especially when large data sets are involved. On the other hand, our technique does not depend on any knowledge of the underlying domain due to our assumption that prior knowledge of attribute correlations is not available or too difficult to obtain satisfactorily.

Automated ranking approach by Agrawal *et al.* [ACDG03] tries to solve the same *Many-Answer problem* as PIRA by making necessary extensions to PIRA's technique which was proposed for solving the *No-Answer problem*. The technique has two variants, *IDF (Inverse Document Frequency) Similarity* and *QF (Query Frequency) Similarity*. Both techniques employ the idea that the more similar a record (viewed as a small document) is to a query (viewed as a set of keywords), the better is the rank. Hence, the ranking function is defined by a similarity measure, where in the case of IDF Similarity is the Cosine Similarity but excluding its normalisation component. In other words, the length of the

artefacts to be matched is ignored, justified by the shortness of the record values used by Agrawal *et al.*. Before the cosine similarity can be calculated, vectors for document, or in this case record, and query must be computed. Agrawal *et al.* have chosen TF-IDF for this task based on the premise that the *Cosine Similarity-TF-IDF* arrangement has been successful in practice. This combination indicates that IDF Similarity takes into account both exact and non-exact matches of terms. An immediate advantage of IDF Similarity is the high recall (i.e., the fraction of relevant data that is retrieved), but at the same time, it may suffer from low precision (i.e., the fraction of retrieved data that are relevant). This undoubtedly gives users a wider insight into possibly-related products, but at the same time subjects users to a flood of most likely unnecessary information. Another challenge for IDF Similarity is the decision on a threshold value to use to define what can be considered as similar. Our technique offers exact matches of a searched word, which filters out a good amount of records from the very start. Also, our technique is free from any need for a similarity threshold that typically would be domain-specific.

Agrawal *et al.* pointed out that IDF Similarity may not work well with categorical numerical data, and thus proposed QF Similarity which, like PIRA, leverages the implicit feedback found in query workloads. To show this drawback, Agrawal *et al.* described the following example.

Example:

> "In a realtor database, more homes are built in recent years such as 2000 and 2001 as compared to earlier years such as 1980 and 1981. Thus, recent years have smaller IDF. Yet the demand for newer homes is usually more than that for older homes".

Based on the presented example, Agrawal *et al.* demonstrated their intuition which assumes there exists more interest in recently-built houses than for earlier ones, leading to the frequency of user queries on houses built in year 2001 to be higher in the workload than for the year 1981. Given the frequency of queries, a similarity value is computed; in this example, records that hold the value 2001 are perceived as being more relevant to the user query, and as a result, receive higher ranking. It could be perceived that the essence of QF is term frequency, save that the frequency is of queries found in the entire workload, and not terms found in a corpus. A shortcoming of QF is when the workload is insufficient

to produce useful frequency. Our technique treats categorical numerical data as text. Exclusively handling categorical numerical data is left for future work. Although we too leverage query workloads, our technique does not rely on large numbers of queries to generate a ranking. Rather, we extract unique terms from each query in the log and ignore any redundancies.

So far, the workings of both IDF Similarity and QF Similarity have been described within the context of solving the No-Answer problem. In the event of the Many-Answer problem, both techniques may at times encounter a problem where multiple records are *'tied'* to one single similarity score and as a result receive some arbitrary ranking order [ACDG03]. A solution [ACDG03] is to *break the ties*. Basically, this process involves differentiating *'tied'* attributes through information solicited from attributes that were not part of a user's query (*missing attributes*). This idea is similar to the *'unspecified attributes'* from Chaudhuri *et al.* [CDHW06]. Ranking now can be reduced to the task of finding suitable weights for the missing attribute values, which defines a value's *global importance*, and finally discriminating between records.

Knowledge about the interrelationship that may exist between attributes requires familiarity with the domain of interest. Automating this is no easy task. From the paper [ACDG03], in order to achieve this, IDF Similarity faces numerous challenges and has been deemed unsuitable. Because of that, our review shifts completely to QF Similarity. Like before, a query workload is used to gain answers for QF Similarity. Computation of global importance is made possible by hinging upon the intuition that positive characteristics such as *important*, *popular* and *desirable* typically would translate into large numbers of queries. From this intuition, if a missing attribute has been identified (e.g., location of a home), different weights are given to its values based on the number of queries which contain the *important* value (e.g., a record that contains "California" is given highest weight if it appears in the most number of queries). While Agrawal *et al.* place substantial attention on the degree of interest that a user may have on specific topics, our research focus is on the values of the actual data and we regard all terms as comparably interesting. A benefit from our strategy is that the *dynamics* of data values are immediately reflected into the ranking. *Dynamics* here refers to the changes made to the existing data value as well as any additions to the collection of values in the database.

3.4 Discussion

Implicit feedback is an interesting prospect to complement explicit feedback. From the review of related works and the technique we proposed, we can see the potential that lies in implicit feedback in providing information that is unobtrusive, may not be feasible to obtain explicitly from users or may be difficult to gather by explicit methods. However, implicit feedback can be susceptible to individual interpretations, relying on intuitions that are domain-specific. We have demonstrated the use of implicit feedback to suggest a user's need for some set of instances. While in this chapter we investigated instance-level manipulation through the use of implicit feedback, in the next chapter we cover our investigation on the use of explicit feedback to directly manipulate such instances. In Chapter 5 we address scalability issues for this proposal.

To present how we catered for scalability in our proposal in the next chapter, we extended this thesis with another chapter.

3.5 Summary and Conclusion

This chapter has presented an approach and empirical evaluations which fulfil the first part of our aim, that is to investigate the use of pay-as-you-go approaches for instance level data integration, specifically identifying relevant instances. We have also presented our first objective (O1) of designing an approach that ranks mappings based on its relevance to a user's information needs. Section 3.1 of this chapter describes our proposed strategy while 3.2 provided details of the evaluation. We have presented the following:

- An approach that identifies instances relevant to a users information need, by using terms commonly used to depict the instances, which can be found in query logs. This answers RQ1 (*how can we design an approach that identifies instances which are relevant to a user's information need in a dataspace*). We view this as a problem of ranking mappings (RH1a), since in a dataspace, records are retrievable through generated mappings. Methods devised by Chaudhuri *et al.* [CDHW06] and Agrawal *et al.* [ACDG03] also produce rankings based on query workloads, but place the produced ranking on each individual record and leverage functional dependencies. The employment of functional dependencies necessitates definitions to be clearly provided, and in certain cases, domain knowledge may be sought. These conditions are not

inherent in our approach since we do not consider functional dependencies. Unlike our approach that views all terms as equally interesting, Agrawal *et al.* highlight specific topics that are potentially interesting to a user. A consequence of this decision is the time taken to reflect on an existing ranking. Our view is terms found in a query's conditional clause recorded in workloads can be a good source to identify a user's information need (RH1b) and can be useful for ranking mappings (RH1c). We leverage on the rarity of terms found in a mapping's extent to indicate the mapping's relevance to the information need of a user (RH1d).

- Two empirical evaluations that allow us to assess our approach. Each answers RQ2 (*how much query logs is needed to produce stable rankings*) and RQ3 (*would the proposed ranking technique be able to track query patterns that are data source-specific i.e., skew*), respectively.
  - The first evaluation is with regards to how quickly stable rankings can be achieved over a sequence of different log sizes. Our results show that our proposed approach can produce stable ranking for encouragingly small log sizes.
  - The second evaluation investigates how the rankings track query patterns that are skewed towards specific sources. The premise is there are patterns used more in certain data sources than others (RH3a). Hence, would the proposed ranking technique be able to reflect this skewed characteristic? Our evaluation was conducted on two real-life life science data sets. From our evaluation, the generated ranking responded satisfactorily to the level of skew inherent in the query logs.

In conclusion, implicit user feedback, especially in the form of queries posted over a dataspace, can be a good source of identifying a user's information need and used in ranking semantic mappings by manner of pay-as-you-go. However, a major challenge of using implicit user feedback for this purpose is in the extent of interpretation which has to be carried out. This interpretation of relating a term with its enclosing conditional clause has been explored in this work for the equality operator. Much work is needed for other operators. This shall be continued in the future.

Chapter 4

Pay-As-You-Go Configuration of Clustering for Instance Integration

This chapter describes the strategy we propose to fulfil our second objective — to devise a technique to integrate instances from heterogeneous and large data sources, while taking advantage of knowledge from users, in the form of feedback, on the domain of interest (O2). This leads to the question of how can we devise a strategy to integrate instances from different large data sources in a dataspace (RQ4). Earlier, we presented an example, showing the integration of instances (Section 1.1). The example showed the primary impediment to obtaining good integration which is the existence of heterogeneity, specifically *data heterogeneity*. *Instance integration in dataspaces*, as we have described in Section 1.2.2 of Chapter 1, is a phase closely related to the querying of results. The aim of our thesis is to investigate the use of pay-as-you-go approaches for instance level data integration. Specifically, this involves identifying relevant instances and identification of duplicates. The product of this phase is a set of relationships, connecting records that are semantically the same. These records are not yet in a state viewable by users since more work needs to be conducted to produce a unique, non-conflicting and clean version. This complicated task is known as *data fusion* and is beyond the scope of our work. We cast instance integration as the problem of *incremental clustering* (RH4a) because knowledge in a domain changes over time. We adopt *pay-as-you-go integration* because users possess valuable domain knowledge that can be useful for improving integration (RH4b), especially when syntactically different words are involved. Furthermore, users are not expected

to identify all semantically similar records at the start, but over a period of time, i.e., the dataspace's life time. We deploy the pay-as-you-go approach to *parameter tuning* because each dataset has its own unique characteristics, and hence differs in the most suitable parameter values (RH4c). The approach is guided by a *user-driven objective function* because different users have different needs for information (RH4d).

We structure this chapter to first provide an introduction to our proposal, where we give the assumptions underlying our proposed technique. We then provide a description of current learning approaches, highlighting the ones that are closest to ours, and a description of the essential steps in instance integration, specifying where user feedback can be useful. In section 4.2, we present some technical context essential to understanding our proposal, followed by the fundamentals that underpin our proposal in section 4.3, i.e., the application of evolutionary search and a user-driven objective function. We address the instance integration problem in sections 4.4 and 4.5 by proposing a baseline in the former section and in the latter four variants of a pay-as-you-go approach to integrating instances. Section 4.6 shows the experiments we conducted to evaluate our proposal and their results. Then, we discuss our findings in section 4.7 and compare our proposal with other work in the following section. Finally, we summarise and conclude in section 4.9.

4.1 Introduction

The objective of this section is to introduce our proposal. We start by providing two types of assumptions that we used to form our technique – system and user-related assumptions. Considering that our proposal hinges upon domain knowledge from users, in the form of feedback, we explored existing approaches of the same spirit i.e., learning-based techniques and underline the one that is nearest to ours: semi-supervised clustering with constraints. This section ends by describing the steps to integrate instances and pinpoints where it can be beneficial to elicit user feedback.

4.1.1 Assumptions

We introduce our proposal by starting with a set of assumptions that underlie our technique. They are divided into system and user-related assumptions.

System-related assumptions:

- The data to be integrated has undergone a preparation process, e.g., date and time formatting and renaming of attributes. Different locales adopt different date and time formats, for example, *dd/mm/yyyy* in the UK and *mm/dd/yyyy* in the US.
- Ontologies of the domains are not available, necessitating dependency on text-based matching schemes. An ontology is known to describe concepts formally existing in a community and their relationships [Gru01]. With ontologies, synonymous words where most text-based matching schemes are unable to find similar concepts, can be identified. For example, the words *dry* and *desiccated*.
- Schema matching and semantic mapping have been conducted to produce a set of semantic correspondences and mappings between local and global schemas.
- There exists a working mapping selection and query component actively deciding on relevant mappings and returning records after query reformulation.

User-related assumptions:

- The user has knowledge of which instance pair points to the same object of the real world.
- Users are expected to provide feedback on only a subset of instance pairs;
- Other than user feedback, the ground truth is absent, hence, no training set in the form of labelled records is present.

4.1.2 Integrating instances

At the very core of instance integration is the task to sort out discrepancy between record pairs identified as duplicates and the real or correct duplicates. A naïve approach to integrating instances is to compare each record with every other record in the dataset to identify any similarities, and afterwards to group similar records together. This is an expensive approach, considering that if there are n records, then $O(n^2)$ comparisons are required.

Much effort has been spent to handle the instance integration problem, particularly by using machine learning approaches to identify matches after learning

from a labelled subset, considering manually identifying all matches is infeasible. Some current learning approaches include the following [EIV07, CSZ06].

Supervised learning relies on training data to *"learn"* how to match records using machine learning techniques which involves training a classifier to distinguish between similar and dissimilar record pairs. Because of its substantial reliance on a training set, the set must provide enough *covering* as well as *challenging training pairs* able to reveal the subtle features of the deduplication function [SB02]. However, putting together such a training set involves tedious manual labour.

Unsupervised learning is an alternative to supervised learning. No set of labelled data is needed with this learning approach. Most unsupervised learning approaches for instance integration adopt the idea of similar vectors corresponding to the same class [EIV07], where each vector represents a record, and each class represents an object in the real world. Hence, it is only natural to have *clusters* of records that refer to the same real world object. This approach depends heavily on generic distance metrics to group together records or separate them.

Semi-supervised learning (SSL) lies between fully supervised learning and learning without supervision. With most data in SSL unlabelled, SSL still enjoys some supervision information although not necessarily for all examples [CSZ06]. Hence, we could divide the data into two groups: points $X_l = \{x_1, ..., x_l\}$, having labels $Y_l = \{y_1, ..., y_l\}$, and points $X_u = \{x_{l+1}, ..., x_{l+u}\}$, possessing labels which are not known. A label signifies that two records point to the same object in reality. Chapelle *et al.* [CSZ06] reported the possibility of SSL assuming other forms, and thus, holding different interpretations. Most of the forms see SSL as supervised learning supported with information about the distribution of examples. This interpretation shares not only its goal with supervised learning, but is also susceptible to the same problem: it is less applicable when the number and nature of the classes are unknown and have to be inferred from data. One particularly interesting form of SSL which is not exposed to such problems is semi-supervised clustering with constraints.

Semi-supervised clustering with constraints [BBBM06]. This approach handles the problem of clustering a set of data points into separate groups with limited supervision in the form of pairwise *constraints*. There are two types of

constraint: *cannot-link* which indicates dissimilarity between instances in a pair, and *must-link* that represents similarity. Unlike labels that require a domain expert and prior knowledge about corner cases in the dataset, constraints can be specified by someone who is not an expert and has no significant prior knowledge of a dataset's classification. Proposed methods for semi-supervised clustering fall into two general types.

1. *Constraint-based* methods use the given supervision information to achieve a set of data partitions inclined towards respecting constraints.

2. Central to *distance-based* approaches, is the employment of a distance function. This function is expected to be parameterised and the corresponding values are *learned* to further separate cannot-link points and increasingly tighten must-link points together.

Our proposal shares several features with *semi-supervised clustering with constraints*: *a*) user feedback carries validation about the similarity of an instance pair; *b*) a set of user feedback is not a set of categorised records which provides covering and challenging record pairs, or corner cases; and *c*) a clustering algorithm is used. A feature that is specific to our current proposal is we do not attempt to mine any *criteria of beneficial or informative records* from users, and have left this for future work.

Active learning is a form of semi-supervised learning. Rather than using a static training set, an active learner selects the most *"informative"* instances for labelling, where an informative instance is one that would provide highest information gain to user. This decision as to which records are most suitable for annotation is performed by an algorithm which is typically based on some set of heuristics, where it is assumed that there is a known and plausibly agreed criteria for *"informative"* instance pair.

Steps to integrate instances Regardless of the type of approach, instance integration can be generalised into three steps. They take place in the following order.

1. **Blocking.** The aim of this process is to reduce the comparison space for instance pairs by limiting comparison to individual blocks. Blocking assumes that similar instances tend to gather in the same block. Generally, the formation of blocks is through the use of a function capable of bringing

out similar features across a set of records, tying similar records together. An example is the use of Soundex on one or more highly discriminative attributes. As a result, a set of small-sized comparison units is produced i.e., *blocks*. We present a formalisation for blocking in subsection 4.2.1.

2. **Pairwise comparison.** Identifying pairs of semantically similar instances is the primary goal for this step. To do this, similarity metrics are used, in combination or alone, to syntactically compare instances. Comparing instances is not trivial. Concerns include the effectiveness of the chosen metric to capture hidden features of instances. We have discussed the issue of effectiveness and presented existing metrics in section 2.4.1. The output of this step is a set of scores that represent the degree of similarity between each pair of instances.

3. **Clustering** uses the similarity scores produced from comparing pairs of instances to group together instances that describe the same object in the real world, and to separate instances that represent different objects. In general, there are two forms of clustering: hierarchical and partitional. Hierarchical clustering performs repeated clustering of instances by splitting and merging clusters to produce a series of nested clusters [Jai10]. Partitional clustering takes each instance and decides its suitable cluster based on how *"agreeable"* the instance is with a cluster [GCBR05]. The output for this step is a set of clusters that contain one or more similar instances, denoting that each cluster represents a single real world object. In subsection 4.2.2 we lay out the formalisation for clustering.

There are many artefacts that when supplied with user feedback can benefit an integration system. Central to instance integration is the artefact that characterises the relationship between two records. A user's knowledge of similar records can be *directly applied* to the integration process [WKFF12, JFH07, JFH08, JSD+13, TMC12] by changing the similarity score of a record pair, or used as an input to a process which performs integration [CR02, **?**, SD06, TKM01, BM03a, BM03b]. Using user feedback in instance integration is beneficial in that the received domain knowledge is used to improve the integration. Unfortunately, acquiring user feedback can be costly, hence, the amount of feedback obtained is best kept at a minimum. In this thesis, we refer to this method of applying user's knowledge simply as *score change (SC)*. An absolute score of 1.0 is given for a matching pair to denote semantically similar records, while a score of 0.0

for an unmatching pair. It is interesting to change scores directly. It could help us to answer the question: If information from user feedback alone is sufficient to improve instance integration where the integration process is viewed as a black box (RQ5).

4.2 Technical Context

We start the presentation of our proposal with a formalisation of important notions and terms to help with the explanation of our proposal later in the chapter.

4.2.1 Blocking

Definition 1. Given a dataset $D = \{d_1, d_2, ..d_M\}$, the aim is to produce a set of blocks $\{B_1, B_2, ..., B_N\}$, where $\bigcup_{i=1}^{N} B_i = D$; subsequently, any instance comparisons are limited to just those instances in a block.

The nature of blocking can be distinguished into two forms: disjoint and non-disjoint blocking. In disjoint blocking, each instance appears in only one block, where

$$\forall_{i,j} B_i \cap B_j = \emptyset \text{ with } i = 1, \ldots, N, j = 1, \ldots, N, \text{ and } i \neq j$$

while in non-disjoint blocking, an instance d_i can appear in multiple blocks, such that

$$\exists_{i,j} B_i \cap B_j \neq \emptyset \text{ with } i = 1, \ldots, N \text{ and } j = 1, \ldots, N$$

and thus gets compared with every instance in each block that d_i resides in.

Definition 2. A transformation function $f(d_k)$ is used to derive an instance's blocking representation so that two instances, d_k and d_l, that share the same representation can be placed in the same block B_i. These representations constitute the set of index keys, Ind, typically used to retrieve V, a set of candidate duplicates of d_k. A widely-used transformation function is the hash function, $Hash(d_k) = h$, where h is a hash key associated with a single block B_i.

4.2.2 Clustering

Clustering characterises a dataset D by separating dissimilar instances and grouping similar ones. A common determinant of this decision is with a measure of similarity between an instance pair. However, the number and choice of instances

to compare is subject to the blocking strategy adopted. We can formalise this as follows.

Definition 3. Given dataset D, a set of clusters $C = \{C_1, C_2, .., C_S\}$ is generated after the similarity of each instance pair (d_k, d_l) residing in each block B_i is calculated. Conditions to be met are,

(i). $C_x \neq \emptyset$ for x = 1,....,S;

(ii). For overlapping clusters, $C_x \cap C_y \neq \emptyset$;
For non-overlapping clusters $C_x \cap C_y = \emptyset$;
where x = 1,..,S; y = 1,...,S and $x \neq y$;

(iii). $\bigcup_{x=1}^{S} C_x = D$

Definition 4. A similarity function $Sim(d_k, d_l)$ which compares two instances, d_k and d_l, should be symmetrical. $Sim(d_k, d_l)$ produces a large value when d_k and d_l are similar, generates its largest value if the instances are identical and has a target range of [0,1].

Often, a similarity function relies on a threshold to distinguish between the notions of similarity and dissimilarity which contributes to the decision to place two instances together in the same cluster or apart.

4.2.3 Evolutionary search

Algorithm 3 An Evolutionary Algorithm in General [ES15]

```
1: INITIALISE population with random candidate solution
2: EVALUATE each individual in population
3: repeat
4:    SELECT parents from population
5:    CROSSOVER pairs of parents to produce offsprings
6:    MUTATE offsprings
7:    JOIN offsprings to generate new population
8:    EVALUATE each individual in new population
9: until TERMINATION CONDITION is fulfilled
```

Individuals as a representation

An evolutionary search starts with the mapping of a real world problem into the space of problem-solving: the encoding of phenotypes into genotypes. *Phenotype* is a term used to refer to an instance of the candidate solution offered for a given problem, and *genotype* refers to the individual algorithmic representation of the

proposed solution, or simply ***individuals***. For example, consider a problem to optimise integers. Hypothetically, to find an optimal solution, it is necessary to convert these integers into binary codes. In this case, the phenotype is identified as a set of integers and the genotype is the representations of the integers in binary. Towards the end of the evolution process, the search finally converges to find an *"optimum"* genotype, though there is no guarantee that this will occur. Depending on where an optimum resides, it can be *local* or *global*. A ***local optimum*** can be found at a local region, while if the coverage is the entire search space then it is recognised as a ***global optimum***. Often, there can be multiple local optima with only a single global optimum.

Forming a population from individuals

Often the initialisation of a population is randomly seeded. With different set of seeds, different individuals are placed together, which eventually would produce different results; hence, the stochastic manner of the evolutionary search. At every generation, a ***population*** is assembled, serving as a placeholder for individuals of some chosen amount. A *"good"* amount of individuals can be difficult to determine. It should be large enough to allow diversity to be included when selecting members of the next population. This amount that translates to the size of the population remains invariable at every generation, throughout the evolution process. A population presents a single unit of evolution — with constant shifting of members — in contrast with the actual individuals that are static in nature.

Parents selection

Selecting parents from an existing population is to meet two purposes: *i*) to open the opportunity for high quality individuals to transfer their good traits to the next generation; and *ii*) to avoid greedy search from occurring, where a search cannot escape a local optimum. Parent selection has the authority to direct the evolution process towards improving. An individual successfully becomes a parent when it has been chosen to be put through several variation operations.

Operations of variation

Variation is applied to individuals found to have properties fit for use across generations. From these operations, new individuals emerge to populate the next generation, amounting to the production of new candidate solutions, if viewed

from the plane of phenotypes. There are two operations that deal with variation: crossover and mutation.

Crossover, a binary operator, accepts a pair of genotypes and merges their eligible traits to form one or more new individuals — their offsprings — in the hope of inheriting only the good traits. Being stochastic, randomness plays a central role in the choice of what traits are combined and how they are combined. It is commonplace for an offspring with combinations of traits less favourable than their parents to occasionally appear. An example [ANd06] of a simple crossover, namely *one-point crossover*, is to randomly choose a point in a genotype to perform the crossover, then copy over from one genotype the part in front of the point to an offspring, and the part behind the point of a second genotype to the same offspring. The symbol `|` indicates the point of crossover.

```
Genotype A:    11111 | 00100110110
Genotype B:    10011 | 11000011110

Offspring A:   11111 | 11000011110
Offspring B:   10011 | 00100110110
```

Mutation, unlike a crossover, is a unary operator. It accepts a single genotype and slightly modifies it to produce an offspring. Like a crossover, mutation is stochastic, suggesting that the formation of its mutant offspring relies on the component of randomness which in turn relies on a specified rate of probability. Typically, a low rate is used because a high probability rate risks the search being catapulted to distant regions, resulting in a greatly modified offspring. This condition is unfavourable if an optimal solution lies within the local region, deeming it to never be found. Nevertheless, escaping a local region can form *"fresher"* individuals if the optimal solution is outside the local region. Depending on the type of data used to represent a candidate solution, the process of mutation is conducted accordingly. For example, if the genotype is a vector of type Boolean, a bit-flip mutation would be suitable [MBWB99, Luk13]. This form of mutation basically moves through the vector and decides based on probability to flip the bit at the current position. If instead the genotype is a float value, one way is to use Gaussian convolution [MBWB99, Luk13]. Fundamentally, each value in the genotype is injected with random noise taken from a Gaussian distribution with a mean $\mu = 0$. Although largely the produced noise would flock around 0, the rare incident of obtaining a big value can occur.

Crepinsek *et al.* [CLM13] conducted a survey on current evolutionary algorithms to investigate their take on the longstanding view, first discussed by Eiben and Schippers [ES98], which suggest that the ***exploration*** of a search space is obtained through *mutation* and *crossover*, whereas ***exploitation*** is achievable through *selection*. Exploitation is an essential process since it is the process of visiting newly explored regions in the vicinity of already found regions. Conversely, Wong [WLLH03] considers exploration is achieved through mutating individuals, and crossover lends a hand in exploiting useful traits of eligible individuals. Despite such differing opinions, these abilities to explore and exploit lay the chance for a global optimum to be found, if enough time is given [Luk13].

The JOIN operation

Together with the variation operations, JOIN makes it possible to distinguish between evolutionary algorithms [Luk13] since the choosing of individuals to continue the lineage happens at this stage: be it a new individual (child) or an old individual (maybe a parent). JOIN is also known as the ***"replacement"*** operator, owing to its task of replacing a population of individuals with their (hopefully) more qualified successors.

Fitness function

This is a function that is used to evaluate the quality of a candidate solution — genotype. Every measured genotype is assigned a value that represents its fitness, or quality, based on a preconceived definition embodied into the fitness function. In other words, its origin is in the phenotype space, but its application is in the genotype space [ES15]. It is good to know that this function not only defines fitness but also the meaning of improvement by creating a benchmark to be achieved [ES15]. Typically, a fitness function promotes to an evolutionary task the reaching of a maximum value (***maximisation***), or a minimum value (***minimisation***). The fitness function is also known as an ***objective function***.

Conditions for termination

Evolutionary searches are stochastic in nature. A guarantee of achieving an optimum fitness level as defined through an objective function or finding an optimum individual is absent. Therefore, several more achievable stopping conditions can be used instead, preventing an endless running of the evolutionary algorithm. Eiben and Schipper [ES15] reported the following stopping conditions.

1. The maximally allowed CPU time elapses;
2. The total number of fitness evaluations reaches a given limit;
3. For a given period of time (i.e., for a number of generations or fitness evaluations), the fitness improvement remains under a threshold value;
4. The population diversity drops under a given threshold.

The approach to use stopping conditions depends on the availability of a specified optimum fitness level. In the case that one exists, a disjunction is used as such — *optimum value reached OR condition x is met*. However, if no optimal is available, any one of the listed conditions can be used.

4.3 Our Approach

Presented in this section are three cornerstones that our approach is founded on. Firstly, the use of user's knowledge to infer similarity between records. Secondly, the manner by which we manipulate an evolutionary search to be applicable to our domain of choice. Finally, the definition of an objective function to guide the evolutionary search.

4.3.1 Inferring user's knowledge on similarity

Users have unique information needs, and only they know if these needs are exactly satisfied. We assume a user of a dataspace has knowledge, to a certain degree, about objects of the real world, and thus is able to specify, to some extent, if a pair of records describes the same object or otherwise. To expect users to manually specify this knowledge on every record pair is typically infeasible, and indeed all explicit feedback-giving can be considered to be costly.

It is useful to be able to propagate the scarce knowledge on similarity learnt from the subset onto the entire dataset. We use a method of inferring whereby a similarity measure is learnt in light of the subset of user feedback, which is used for clustering the rest of the dataset in the same manner. Parameterisation is installed to allow this learning. An evolutionary search is used, which explores a search space through guidance by user feedback, to find a set of parameters (at this point, weights) which nears an optimum.

We reuse our example in Figure 1.1 to show how weights on attributes can help to discriminate instances. Weights are a simple yet powerful method to distinguish instances based on attribute importance as perceived by users. Assume

we have two records, *CS1* and *CS2* that are undergoing deduplication. A typical text matching algorithm may find the two records to be very similar, however, a user may know otherwise and indicates the discriminative property of the *"date of birth"* attribute with weights. As a result, these records are identified as non-duplicates and are separated into different clusters. In the case that record *45* is included in the deduplication process, assuming it has undergone data preparation, the weight on attribute *"date of birth"* would highlight its similarity with record *CS1*, resulting in the pair of records being clustered together.

Local schema 1: Customer

ID	FirstName	LastName	***DateOfBirth***	Block	Level	Unit	Street	Postcode
CS1	John	Smith	***3/6/75***	17	2	6	Hartley Road	M168PA
CS2	John	Smith	***6/3/75***	18	2	6	Hartley Road	M168PA

Local schema 2: Buyer

ID	Name	***DOB***	Lot	Street	Postcode	Country
45	John B. Smith	***3 June 75***	17-02-06	Hartley Rd.	M16 8PA	UK

Based on a global schema: Cust

Cluster 1:

ID	FirstName	LastName	Block	Level	Unit	Street	Postcode	Country
CS1	John	Smith	17	2	6	Hartley Road	M168PA	
45	John B.	Smith	17	2	6	Hartley Road	M168PA	UK

Cluster 2:

ID	FirstName	LastName	Block	Level	Unit	Street	Postcode	Country
CS2	John	Smith	18	2	6	Hartley Road	M168PA	

Figure 4.1: Formed clusters when weight is given on "Date of Birth" attribute

In this exercise, for simplicity, we regard number values as strings and demonstrate the use of a single similarity measure, n-gram (Equation 4.2). This practice of considering number values as strings is not especially rare as stated in [EIV07].

In our approach, we define similarity as a degree of match between every semantically-corresponding attribute of two records. We decided that comparing every attribute with every other is unnecessary in our case due to the following reasons.

1. We made the assumption that the matchings between schemas are correct: mappings involving *one-to-one (i.e., a single attribute maps to another single attribute)* and *one-to-many (i.e., a single attribute maps over to multiple attributes)* associations have been appropriately managed.

2. Our datasets largely contain attributes that carry discrete information e.g., the attributes {*id, name, description, price*}. Although *name* and *description* may share some common textual description, but the number are small

if compared with the number of attribute permutations that do not have overlapping text.

3. The information on attributes with overlapping contents is not available.

4. Weights specify the degree of importance an attribute has when discriminating instances. They are attribute-specific. Comparatively, cross-attribute comparison would suggest a different meaning to the weights.

Consider two records *r1* and *r2* with a set of attributes $\{ar1_1, ar1_2, \ldots, ar1_N\}$ and $\{ar2_1, ar2_2, \ldots, ar2_N\}$ respectively. The similarity between $r1$ and $r2$ can be obtained through the total match value of their corresponding attributes over the number of attributes (Equation 4.1).

$$similarity(r1, r2) = \frac{\sum_{i=1}^{N}(match(ar1_i, ar2_i))}{N} \tag{4.1}$$

Match is a function that divides every text into grams and uses the *Dice coefficient* [Kon05] to calculate the similarity (Equation 4.2). The Dice coefficient highlights the portion of n-grams that overlap between two text strings X and Y over all n-grams generated from X and Y, where $ar1_i$ contains X while $ar2_i$ contains Y.

$$match(X, Y) = \frac{2 \times |ngrams(X) \cap ngrams(Y)|}{|ngrams(X)| + |ngrams(Y)|} \tag{4.2}$$

Weights can be easily incorporated into the similarity definition (Equation 4.3). Weights are specific to each attribute, hence weight w_i applies to only the i-th attribute. The larger the value of the weight, the more important is its role in distinguishing instances.

$$weightedSimilarity(r1, r2) = \frac{\sum_{i=1}^{N}(w_i \times (match(ar1_i, ar2_i)))}{N} \tag{4.3}$$

It is good to note that changing an attribute's weight value can initiate the movement of instances from one cluster to another. We use our previous example (Figure 4.1) to show that changing of attribute weights can influence the membership of clusters. In Figure 4.2, we illustrate this by transferring the weight from the *"date of birth"* attribute to the *"last name"* attribute of local schema 1 and *"name"* attribute for local schema 2, considering this *"one-to-many"* relationship has been handled and that *"Smith"* was successfully extracted. The result is that record $CS2$ no longer appears in a singleton cluster but joins records *CS1* and

45 in their cluster, leaving only one cluster in the set of clusters. This suggests weight-changing can be employed to find *"optimal"* clusters. A set of clusters is optimal when similar records are grouped together and dissimilar records are separated apart, based on some criteria. Finding optimal clusters requires numerous tries with different combinations of weight set. Evolutionary search is suitable for such a task. We discuss the application of evolutionary search in the next subsection.

Local schema 1: Customer

ID	FirstName	***LastName***	DateOfBirth	Block	Level	Unit	Street	Postcode
CS1	John	***Smith***	3/6/75	17	2	6	Hartley Road	M168PA
CS2	John	***Smith***	6/3/75	18	2	6	Hartley Road	M168PA

Local schema 2: Buyer

ID	***Name***	DOB	Lot	Street	Postcode	Country
45	***John B. Smith***	3 June 75	17-02-06	Hartley Rd.	M16 8PA	UK

Based on a global schema: Cust
Cluster 1:

ID	FirstName	LastName	Block	Level	Unit	Street	Postcode	Country
CS1	John	Smith	17	2	6	Hartley Road	M168PA	
45	John B.	Smith	17	2	6	Hartley Road	M168PA	UK
CS2	John	Smith	18	2	6	Hartley Road	M168PA	

Figure 4.2: Cluster set after weight is transferred to "Last name" and "Name" attributes

4.3.2 Applying evolutionary search

We start this subsection by justifying our use of evolutionary search in the context of our problem, then describe how we apply it as a solution.

Depending on the number of attributes chosen for data integration and the amount of parameters used for resolving duplicates, the space for parameter combination in instance integration can be wide. In our setting, relevant parameters include weights for attributes and configurations for a clustering algorithm. Each candidate solution represents one parameter combination.

An evolutionary search is able to support our *"discrete and differently typed"* parameters. At the coarser level, we host two distinct and semantically unrelated groups of parameters. The first group consists of attribute weights while the second holds configurations for the underlying clustering algorithm. Moving to the finer level, each group has a value set which is a mix of integer and float (shown in Table 4.1 on page 92).

The generational nature of evolutionary search permits the notion of *survival of the fittest* where good candidates are given a higher chance to spawn the next

generation of candidates than the less fit ones. Through this iterative process of evolution, beneficial variation is gathered and made more pronounced by means of *"trial and error"* [Bro11].

In a search space, closely-located candidates tend to have near-to-similar traits. A search which concentrates at one particular area would eventually offer *"stale"* candidates. Through *exploration*, distinct regions of the search space can be visited by making small changes to every search point and by combining properties of two or more search points [Cor14]. Visiting closely-located candidates is not always a disadvantage: in the case that a *"good"* candidate is found, finding a *"better"* candidate could simply mean to *exploit* such available proximity.

Evolutionary search offers *independent processing* of individuals of a population. With individuals able to undergo processing simultaneously, an evolutionary search can quickly traverse through the search space to find an acceptable result.

We describe here how the components of the evolutionary search (i.e., GA) are applied to our proposal. We present,

- **a genotype**, the basic unit of searching through evolution, to consist of values from multiple data types. Our genotype represents a single parameter combination. Table 4.1 lists the parameters used for instance integration. It includes weights and configuration for the clustering algorithm we employed. Further explanation on the role of these parameters will be covered in the next section.

Table 4.1: Parameters for instance integration

Parameter	**Description**	**Type**	**Optimisation Range**
$numKeys$	Number of keys per tuple in HASH	Integer	$1 \leq numKeys \leq 9$
$keyComponents$	Number of values contributing to a key in HASH	Integer	N/A – $keyComponents = 1$
q	Size of q-gram in HASH	Integer	N/A – $q = 3$
k	Nearest neighbours returned by kNEARESTNEIGHBOURS	Integer	$1 \leq k \leq 20$
$similarityThreshold$	Maximum distance between candidate duplicates in kNEARESTNEIGHBOURS	Float	$0 \leq similarityThreshold \leq 1.0$
$membershipThreshold$	Voting threshold in MOSTLIKELYCLUSTER	Float	$0 \leq membershipThreshold \leq 1.0$
w_i	Attribute weights in DISTANCE, such that there is one weight per attribute	Float	$0 \leq w_i \leq 1.0$

- **a population** to hold a group of genotypes, in other words a collection of parameter combinations;
- **a selection** strategy taken from a popularly-used technique in GA, the Tournament Selection [ES15, Luk13, MG95]. This technique works by running λ number of *"tournaments"* to select λ members for breeding. Each tournament typically has a size of 2 [ES15, Luk13, MG95]. A larger size would indicate a greater possibility that there will be highly fit members selected, removing more low-fitness members, thus showing a behaviour that is more *deterministic* and less *stochastic* [ES15, Luk13]. Competing members are randomly picked and their fitness compared. The fitter member will *"win"* the tournament and will have the chance to mate.

  Using tournament selection has the benefits of *a*) simplicity, hence can be easily applied [MG95, ES15], *b*) suitability for use when processing occurs in parallel or otherwise [MG95], and *c*) support for *selection pressure* [MG95, ES15], to fine-tune in accordance with a domain's needs.

  An increase in the selection pressure means an introduction of higher competition into the next generation, pressing the evolution to reach better fitness faster, and as a result, quicker *convergence* with the risk of being premature. Convergence is an event where a (potentially local) optimum solution is found. Selection pressure is boosted by inflating the tournament size.

- **a crossover** approach which takes into consideration a multi-type genotype. Existing crossover strategies are *one-point*, *two-points* and *uniform* crossovers. We have described *one-point crossover* earlier. Different from a one-point crossover, which chooses one same point in two genotypes and swapping their genes located after that point, a two-point crossover selects two positions. The genes between this position pair are swapped to form an offspring. Uniform crossover crosses genes at *"each point independently of one another"* [Luk13]. Every gene bears the chance to be selected for crossover. Imagine visiting every gene in the genotype, and at every position stopping to toss a coin. If heads appears, the gene at the current position is swapped. Such a strategy agrees with our discrete, multitype parameters. We use uniform crossover at two levels: *genotype* and *gene*. Crossover at the genotype has the role of determining which of the parameters will be chosen for the task. The actual crossover process occurs at the gene. By limiting crossover to happen within a group that contains the same representation, for example all weights of genotype A and B, or groups of numbers that share the same range, in this case similarity and voting thresholds, the meaning behind every parameter is preserved.

- **a mutation** approach that relies on the type of data used by the genotype. With only integers and floats used, our mutation utilises a *uniform mutation* for integers and *Gaussian mutation* for floats. Basically, a *uniform mutation* randomly chooses a value within a pre-specified number range and assigns it to the gene [Luk13]. Earlier, we have described *Gaussian mutation* in section 4.2.3. For parameters that represent threshold values (*i.e., similarity threshold, voting threshold*), we specified a probability rate of 0.05, intended for combing closeby areas. This exercise is expected to result in the finding of a *"just enough"* threshold value. However, with weights-filled parameters, a rate of 0.1 is given. The intention is for each parameter to explore various distant regions. The intuition is a set of attributes with near similar weights would not be very useful.

- a **JOIN operation** of new individuals with the addition of elites. Elitism is a strategy that includes fittest individuals from the most recent population, namely *elites*, into the next generation [Luk13]. Its aim is to ensure the continuation of good quality individuals throughout the evolution process [BC95]. When preparing a new generation, we replace the entire population minus a specified number with new individuals, and invite the same number

of elites to join.

- two **conditions for terminating an evolutionary search**. In the first condition, if a candidate solution reaches a desired fitness level. In other words an ideal individual has been found — in our case, the maximisation of our objective function. In the event that no solution fulfils the expected quality, the search resorts to a second condition, which is the maximum number of generations to be produced.

  This scheme is commonly used in conjunction with age-based and stochastic fitness-based replacement schemes, to prevent the loss of the current fittest member of the population. In essence a trace is kept of the current fittest member, and it is always kept in the population. Thus if it is chosen in the group to be replaced, and none of the offspring being inserted into the population has equal or better fitness, then it is kept and one of the offspring is discarded.

4.3.3 Objective function

We dedicate this subsection to explaining our objective function. This objective function is driven by the assumption that fitness should be expressed from the user's point of view. Different users have different information needs, and it is not always easy to accurately produce the desired information. Hence, it is a good idea to *guide* the result-searching process from as early as possible down a path that a user prefers.

At the core of our objective function (Equation 4.4) is the feedback given by the user on the similarity and dissimilarity of a subset of instances. The aim of this function is to indicate to the evolutionary process what *portion* from the present cluster set corresponds to what the user knows as duplicate instances and non-duplicate instances. The numerator of the objective function describes the total number of instance pairs *"correctly"* clustered, be they placed together or separated apart. *"Correctness"* depends on the user's knowledge of the actual object(s) that the instance pair represents. The denominator is the sum of all located instance pairs in the cluster set, both correctly clustered or otherwise.

$$\frac{MM + UU}{MM + MU + UU + UM} \tag{4.4}$$

- MM is the number of matched instance pairs which user also views as a match.

- UU is the number of unmatched instance pairs which agrees with the user's view of unmatching pairs.
- MU is the number of matched instance pairs which disagrees with the user's view.
- UM is the number of unmatched instance pairs which the user views as a match.

Here, we have described our implementation of evolutionary search to suit our context. In the next two sections, we describe a baseline technique we have identified and our proposed variants of instance integration through pay-as-you-go means.

4.4 Identifying a Baseline

Before we could commence with our experiments, we need a baseline to provide a foundation on which we can build our strategies. Our selection of a baseline is based on a set of criteria which can be applicable to any proposal aiming to integrate instances by pay-as-you-go. To integrate instances by *pay-as-you-go*, our baseline should be able to:

- handle *incrementality*. Incrementality is essential in handling the introduction of new instances without redoing previous integration effort. With regard to our proposal, incrementality should be over a clustering algorithm.
- manage the $O(n^2)$ *condition*. With the abundance of records in public datasets, the time taken for integrating two sets of data can be enormous. For n records, matching every pair of records equals to $O(n^2)$ comparisons. It is vital for our baseline to adopt strategies to mitigate this problem.
- adaptable to the insertion of *weights*. Since our proposal leverages distinguishing attributes through the use of weights, our baseline should be easily adaptable to the inclusion of weights.

We have found an incremental clustering algorithm proposed by Costa *et al.* [CMO10] to meet our criteria as a baseline (algorithm 4). We refer to this algorithm as ICA for short. ICA is incremental because of its ability to handle instances new to a dataset (line 5). Besides that, ICA does not remove any instances from a cluster but instead adds them hence preserving old integrations (lines 8 to 13).

Algorithm 4 Incremental clustering algorithm [CMO10]

```
1: Input: C Clusters
2: Input: I Index
3: Input: NewTuples Set<Tuple>
4: Input: P Parameters
5: for t ∈ NewTuples do
6:    Nn = get set of nearest neighbours for t based on I and P
7:    c = find most suitable cluster for t from C, depending on Nn, based on P
8:    if c == null then
9:       newCluster = create a new cluster that contains t
10:      append newCluster to C
11:   else
12:      append t to c
13:   end if
14: end for
```

To enable scalability, ICA resorted to the use of a two-level hashing strategy for blocking, utilising *min-hashing* [Ind01]. In principle, min-hashing utilises an overlap formed between two similar records by projecting their common "features" exhibited as short integers. Unlike traditional hashing, min-hashing hosts a set of hash functions known as a hash family and transforms a record using each family member into an array of "representing" short integers. ICA takes it further with a second level of hashing at which the short integer representatives are themselves hashed. Similarly, the smallest hash numbers are chosen and permanently appointed as keys in an index. A feature is basically a 3-gram of the words contained in an instance with its attribute fields ignored. In other words, an instance is regarded as a single long string. Costa *et al.* reasoned that with each instance mapped to a key, finding instances with high syntactic similarity can be contained to the collection of instances that share the same key, which are referred to as *neighbours*. This reduces the number of pairwise comparisons to be conducted.

We present an example to demonstrate the principles of min-hashing behind its application. Given, a record $u = \{Jeff,\ Lynch\}$ and its extracted features, $\{Jef,\ eff,\ Lyn,\ ync,\ nch\}$. Using a family of hash functions, hash codes are generated for each feature. In this example, we illustrate using only two hash families (Table 4.2). Selecting an array of representative hash codes for record u involves, identifying the smallest-valued hash code for each word from every hash family. Here, from the first hash family, 54 and 19 were chosen to represent the words *John Lynch*. From the second hash family, 42 and 31 were chosen, resulting in the following representatives.

$$Representatives_1 = \{\ 54,\ 19\ \}$$

$$Representatives_2 = \{\ 42,\ 31\ \}$$

Table 4.2: Hash codes generated from two hash families for every feature.

	Jef	***eff***	***Lyn***	***ync***	***nch***
Hash family 1	76	54	19	112	201
Hash family 2	81	42	99	98	31

Index key formation can be inclined either towards blocking for precision or for recall. Precision is the fraction of returned instances that are correct; whilst recall describes the fraction of correct instances that are returned. In Table 4.1 the corresponding variables are *numKeys* and *keyComponents*. In principle, a higher value of *numKeys* reflects a larger recall, and higher precision comes from a larger valued *keyComponents*. Determining their values is a matter of realising the characteristics of individual datasets.

Clustering in ICA involves the interplay of three components: a function that calculates similarity, the selection of nearest neighbours based on a value k, and the promotion of a most suitable cluster according to the voting of these neighbours. When a new instance t is available, a set of k of its nearest neighbours (Nn) is established from the hash index (line 6). This decision is made by invoking a function to assign a similarity value between t and every one of its neighbours. Costa *et al.* suggested a similarity threshold of *0.8* and a maximum nearest neighbours (k) of *10*. After some test runs with our dataset, we identified that the suggested threshold was too high to result in any neighbours for certain instances, instead, a threshold of *0.25* produces a more substantial amount. We allow up to *10* nearest neighbours to be gathered for t. Our goal is to permit the establishment of enough neighbours for the clustering to reach its voting stage with the individuals required to obtain good results.

Voting takes place with the single aim of finding a most suitable cluster for t (line 7). Once we arrive at this stage, each neighbour votes for its containing cluster using a normalised similarity measure (Equation 4.5),

$$votingScore(t, nn) = \frac{1}{1 - similarity(t, nn)} \tag{4.5}$$

with $nn \in Nn$ and $similarityThreshold = 0.25$ (see Table 4.1 for the list of parameters). To compare which cluster from set C of all clusters receives the most votes, the score of a cluster c is the sum of the voting score of all t's

neighbours in cluster c (Equation 4.6).

$$Score_c = \sum votingScore(t, nn_i) \tag{4.6}$$

given $nn_i \in Nn \wedge nn_i \in c$; $D = \{\text{all neighbours of } t \text{ in } c\}$; $i = 1, \ldots, |D|$. ICA uses a threshold for voting (i.e., *membershipThreshold*) at *0.5*. According to Costa *et al.* the use of values other than *0.5* for *membershipThreshold* could cause ICA to become inclined towards creating new clusters as opposed to finding the most suitable cluster from the existing cluster pool. Two conditions must be met to qualify a cluster to be the most suitable. First, a cluster must have the highest voting score and second, it must be above the specified *membershipThreshold*.

ICA is agnostic to the use of weights on attributes. Therefore, we have installed this function ourselves. Earlier, we have illustrated how weights can be a simple yet powerful means to distinguish which attributes can help to group or separate instances. Considering the assignment of weights requires some understanding of the real world, our baseline is created by a human in order to make comparison with our variants possible. We list the schema of each experimental data set and its manually-specified weights in Table 4.3. AbtBuy[1] and AmazonGoogle[2] contain information of e-commerce products while DblpAcm[3]) holds bibliographic records.

Table 4.3: Human-specified weights on each dataset's schema

Dataset	**Attributes with human-specified weights**
AbtBuy	{ product_name: 0.6; description: 0.8; price: 0.2 }
AmazonGoogle	{ product_name: 0.8; description: 0.4; manufacturer: 0.6; price: 0.2 }
DblpAcm	{ title: 0.8; authors: 0.4; venue: 0.6; year: 0.7 }

From a bird's eye view, the baseline goes through a process flow as shown in Figure C.1 (Appendix C), starting from its invocation and finally to its evaluation. Configuration parameters used were discussed throughout this section, which are,

- weights.
- similarity threshold.
- voting threshold.
- maximum number of nearest neighbours.
- length of gram.

[1] dbs.uni-leipzig.de/file/Abt-Buy.zip
[2] dbs.uni-leipzig.de/file/Amazon-GoogleProducts.zip
[3] dbs.uni-leipzig.de/file/DBLP-ACM.zip

- a set of hash families.

In a sensitivity analysis across the data sets, three of the parameters show a constant incline towards specific values when best results are of concern. They are $q = 3$, $numKeys = 9$ and $keyComponents = 1$. Due to this, these parameters have been excluded from any optimisation exercises. Once all instances are clustered, the fitness of the obtained set of clusters is measured. The measurement which involves ground truth is detailed in Section 4.6.

4.5 Proposed Variants of Pay-as-you-go Instance Integration

Our variants are extensions of the baseline. They exhibit one or more of the following dimensions:

1. the adoption of optimisation,
2. the configuration parameters that are subject to optimisation,
3. the direct changing of similarity scores by user feedback.

For variants which observe optimisation with configuration parameters, an evolutionary search is implemented to find a favourable set of parameters. Its implementation using parallelism is explained in detail in the next chapter.

4.5.1 No-optimisation score change (NOSC)

This variant, abbreviated as NOSC for *No Optimisation Score Change* (Figure D.1 in Appendix D), directly applies the user's knowledge about objects in reality onto relevant record pairs. We avoid any optimisation for this variant in order to form an understanding of the benefit that results from *directly changing* the associations among records. NOSC adopts the same configuration parameters as our baseline.

4.5.2 Weight-only Optimisation (WOO)

WOO investigates optimisation through an evolutionary search, specifically using a genetic algorithm (GA). The only configuration parameters fed into WOO for optimisation are weights (see Figure E.1 of Appendix E). For every set of weights proposed by the optimiser, all records are clustered and the resulting cluster set

is evaluated using the fitness function described in section 4.3.3. This process is iterated a number of times (red loop in figure E.1), translating into the number of *generations* to complete in the GA.

4.5.3 Weights-and-parameters Optimisation (WAPO)

Extending from WOO, WAPO optimises not only weights but all the configuration parameters needed by ICA. Innate to every dataset is its individual characteristics, therefore, it can be expected that to assume constant parameters and weights to apply to all datasets is unsuitable. Figure F.1 in Appendix F illustrates the flow of processes in WAPO.

4.5.4 Post-optimisation score change (POSC)

Our final variant tests an alternative path to determine if directly changing scores can have some benefits on integration (refer Figure G.1 in Appendix G). This is accomplished by conducting direct changes after optimisation has completed. The intuition is if direct-score changing is effective, then there would be areas of the integration landscape that would exclusively gain from it and not from optimisation alone. Hence, POSC should produce fitter clusters than WAPO. In Figure G.1 we show where in the process flow direct-score changing occurs. We divide POSC into two phases. The first aims at finding an optimised set of weights and parameters, which executes as WAPO. In the second phase, we reuse NOSC to cluster records by direct score changing. Then, the produced cluster set is evaluated against the ground truth.

We summarise in Table 4.4 the dimensions adopted by each of our variants.

Table 4.4: A summary of dimensions in variants.

Variants	**Optimisation**	**Parameters optimised**	**Score change**
Baseline	No	N/A	No
NOSC	No	N/A	Yes
WOO	Yes	Weights only	No
WAPO	Yes	All but q, $numKeys$, $keyComponents$	No
POSC	Yes	All but q, $numKeys$, $keyComponents$	Yes

4.6 Experiment

We seek to understand what is the effect of user feedback on improving instance integration when the integration is viewed as a black box (RQ5) and to answer the question of how effective is our approaches in integrating instances at different levels of user feedback (RQ6). To obtain this, we conducted two experiments which test on different amounts of feedback from users. Prior to this pair of experiments, we conducted a series of runs for two of our data sets using WAPO: AbtBuy and AmazonGoogle. The purpose of these runs is to test on the consistency of the produced fitness values when different random seeds are used. We use WAPO to execute the runs considering that it possesses all that we hypothesised – incremental clustering, evolutionary search, optimisation of attribute weights and parameters of the underlying clustering algorithm and takes advantage of user's knowledge of a domain. Further details of the tests are presented in Appendix H. In summary, the result showed a consistent level of fitness across five runs, using five unique sets of random seeds with a feedback amount of 250. From this test, we learnt that even with random sets of seeds our technique produces values that centre around a single point. Therefore, for the following experiments, we ran each variant once for every data set.

4.6.1 Experimental Setup

Dataset

All three datasets (AbtBuy, AmazonGoogle and DblpAcm) contain real data and have been made available by the University of Leipzig. We have found other studies [WKFF12, LLH11, KTR10] to have also used them for instance integration investigations. We list the details of these datasets in the Tables 4.5, 4.6 and 4.7.

.

Evolutionary search

We adopted the *ECJ evolutionary computing system*[4] to execute our evolutionary search. In the light of experiments with various values, we came to learn that a population size of 50 and a generation size of 70 produced comparably acceptable and stable results as when more searching is done. This is not surprising considering a population size as small as 30 has been shown through empirical studies to be quite adequate in many cases [SCED89, Gre86].

[4] cs.gmu.edu/~eclab/projects/ecj/

Table 4.5: AbtBuy dataset

Attributes	{ product_name, description, price }
E.g., record	*6493, "Denon Stereo Tuner - TU1500RD", "Denon Stereo Tuner - TU1500RD/RDS Radio Data System AM-FM 40 Station Random Memory Rotary Tuning Knob Dot Matrix FL Display Optional Remote", $375.00*
Total records	2173
Total record pairs	2,359,878
Total duplicates	1097

Table 4.6: AmazonGoogle dataset

Attributes	{ product_name, description, manufacturer, price }
E.g., record	*http:www.google.combasefeedssnippets12244614697089679523, "production prem cs3 mac upgrad,adobe cs3 production premium mac upgrade from production studio premium or standard", "adobe software", 805.99*
Total records	4589
Total record pairs	10,527,166
Total duplicates	1300

Table 4.7: DblpAcm dataset

Attributes	{ title, authors, venue, year }
E.g., record	*672969, "An Effective Deductive Object-Oriented Database Through Language Integration", "Maria L. Barja, Norman W. Paton, Alvaro A. A. Fernandes, M. Howard Williams, Andrew Dinn", "Very Large Data Bases", 1994*
Total records	4910
Total record pairs	12,051,595
Total duplicates	1083

Feedback generation and ground truth

We view feedback as a subset of the ground truth. In practice, the portion of *non-match* record pairs is often very much larger than *match* pairs [WLYF11, WKFF12, GC06]. We distinguish between unmatch and non-match pairs: the former describes two records that hold the possibility to be semantically similar but were not identified to be so; the latter holds a more definitive position of two records being semantically dissimilar. Being publicly available for duplicate detection studies, all three datasets were provided with predetermined *match* pairs. Our task now is to generate ground truth and choose from it a group of match and unmatch pairs that comply with the feedback size requested. Considering there is no evidence that users have any tendencies to prefer annotating matches over unmatches, our experiments allocate equal numbers of both feedback types

in every batch. We use the following formula to produce the total record pairs for R records which also represents the population of the ground truth for the data set in question (Equation 4.7).

$$\frac{R \times (R-1)}{2} \tag{4.7}$$

Each pair is then labelled based on its type. Then, we randomly choose from the collection of *match* ground truth a subset for feedback. Since total match ground truth is inherently small in size, randomly selecting pairs seems unproblematic. However, this is not the case for unmatch ground truth. Because of its magnitude, we have to be more selective. Hence, it is more advantageous to select syntactically similar but unmatched record pairs than obviously dissimilar ones. Unmatch feedback is generated by: *a*) executing a blocking algorithm and pairing up records that share the same block and *b*) randomly selecting pairs that are specified as unmatch in the ground truth.

The fitness of a cluster set can be determined by comparing it with the ground truth. We define fitness of a cluster set in regard to ground truth (Equation 4.8) as,

$$(\frac{M_G M}{M_G M + M_G U} + \frac{U_G U}{U_G U + U_G M}) \times \frac{1}{2} \tag{4.8}$$

where

- $M_G M$ represents the number of record pairs found to match in the cluster set and that also match in the ground truth,
- $U_G U$ indicates the total number of unmatch record pairs that the cluster set has where ground truth sets them as unmatch as well,
- $M_G U$ is the number of record pairs which the final clusters identify as a match but the ground truth perceives otherwise,
- $U_G M$ is the number of record pairs in a cluster set that are not duplicates but where the ground truth distinguishes them as duplicates,

The fitness measure makes use of the fractions of all record pairs in agreement with the ground truth for both match and unmatch entries. The fitness measure gives equal weight to the evidence from matched and from unmatched entries, to avoid the risk that the effect of matching entries is swamped by the number of unmatches, which is often a great deal more than matches.

In the following subsections we describe our experiment design and share the results generated. These experiments are based on this definition of fitness. In both our experiments, at the point where feedback is not available, this is analogous to a run of our *Baseline*.

4.6.2 Experiment 1: Compare Proposed Strategies with Three Feedback Levels

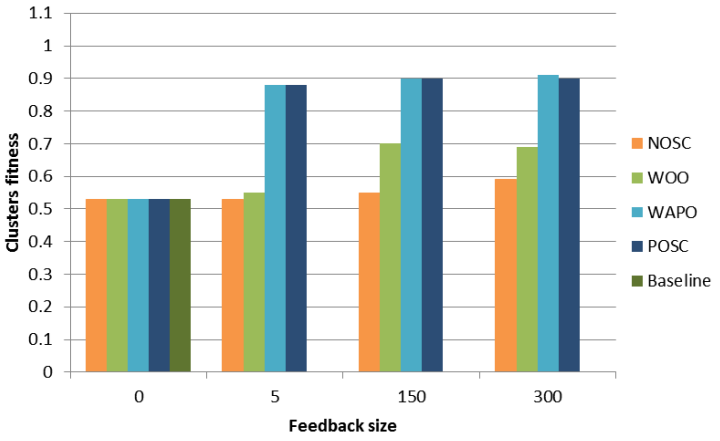
This experiment has the objective of understanding how our proposed variants would respond when feedback at three different levels (i.e., 5, 150, 300) is introduced. Knowing this would allow our subsequent experiment to execute with the most encouraging variants. The smallest level of feedback is 5, which translates into less than 1% of every data set. 300 is the largest level and it represents 6% of AmazonGoogle and DblpAcm each, whereas for AbtBuy, this level constitutes 14% of its total records. Table 4.9 displays the different levels of feedback and the percentage they represent for each data set.

Table 4.8: Feedback sizes and percentages across data sets

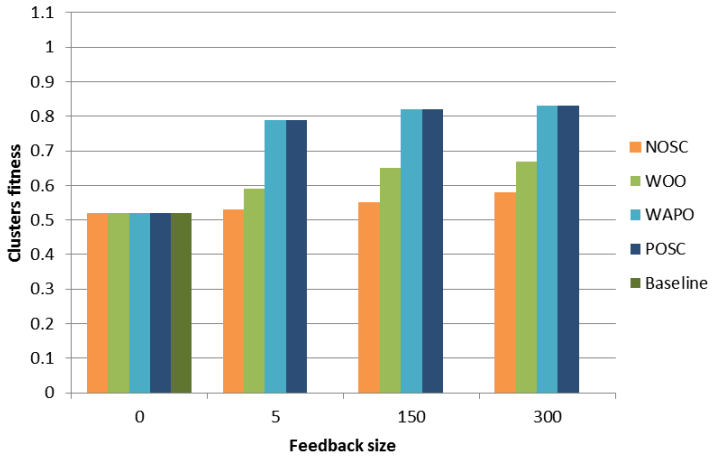
Data set	**Total records**	**Feedback**	**%**
AbtBuy	2,173	5	<1
		150	6.9
		300	14
AmazonGoogle	4,589	5	<1
		150	3.3
		300	6.5
DblpAcm	4,910	5	<1
		150	3.0
		300	6.1

We expect some performance improvement with the presentation of feedback. Our experiment yields the results in Figure 4.3. The following can be observed:

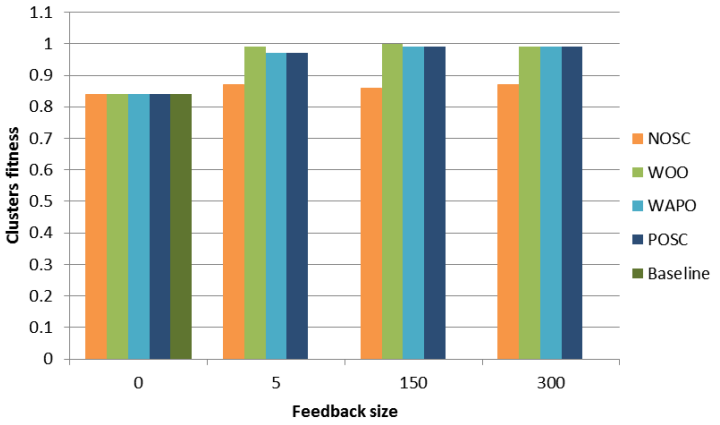
- In the absence of any feedback, all variants show comparable cluster fitness with the baseline; however, improvement in the level of fitness is obtained when feedback is presented.
- The direct application of user feedback (*score change*), as could be observed in NOSC, produces limited improvements regardless of the amount of feedback introduced.
- In contrast, when user feedback is reflected in the weights and parameters of the algorithm using WAPO or POSC, we could observe improvement in the performance even with small amounts of feedback. Although this is less apparent in WOO when AmazonGoogle and AbtBuy are used, this is attributed



(a) AbtBuy



(b) AmzonGoogle



(c) DblpAcm

Figure 4.3: Experiment 1 – Clustering fitness over three feedback sizes

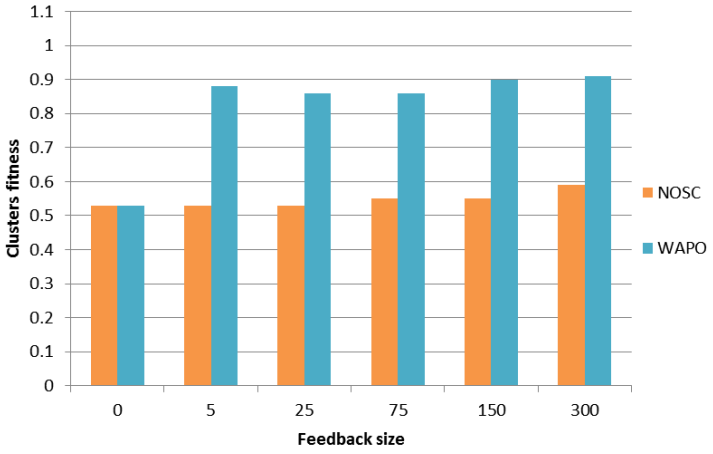
to the higher complexity of these datasets as compared to DblpAcm. Both AmazonGoogle and AbtBuy have long text embedded with descriptions that do not overlap, while text in DblpAcm is shorter and permuted.

- Besides for DblpAcm, WOO consistently shows lower performance than WAPO, indicating that in cases where the dataset is more *"varied"* [IRV13] (e.g., contains permutation, under-specified words or synonyms) it may be beneficial to include configuration parameters into the optimisation process.
- At feedback level 150, the performance of all optimisation-based strategies (WOO, WAPO and POSC) has levelled off, showing no further changes in the fitness.
- POSC exhibits very similar performance to WAPO, implying WAPO's capability to cluster data items pegged with feedback correctly in general, even in conditions void of any evidence from changing of scores.

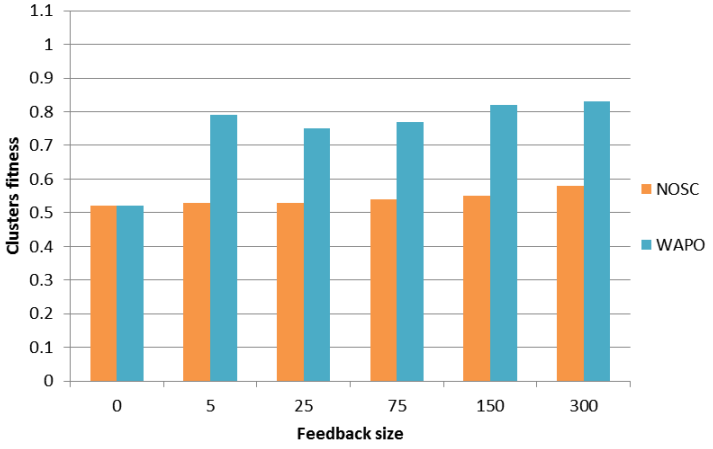
4.6.3 Experiment 2: Compare Selected Strategies with Varying Feedback Amounts

Extending Experiment 1, this experiment runs our promising variant, WAPO, against a set of feedback with smaller increments (i.e., 5, 25, 75, 150 and 300). These finer intervals of feedback size allow us to know more precisely the rate by which the quality of a clustering can be expected to improve with the growth in feedback. We chose WAPO for its encouraging results. However, we have included NOSC into this experiment for the sole purpose of comparison, serving as the variant which feedback is introduced with the absence of any optimisation. The results can be observed in table 4.4.

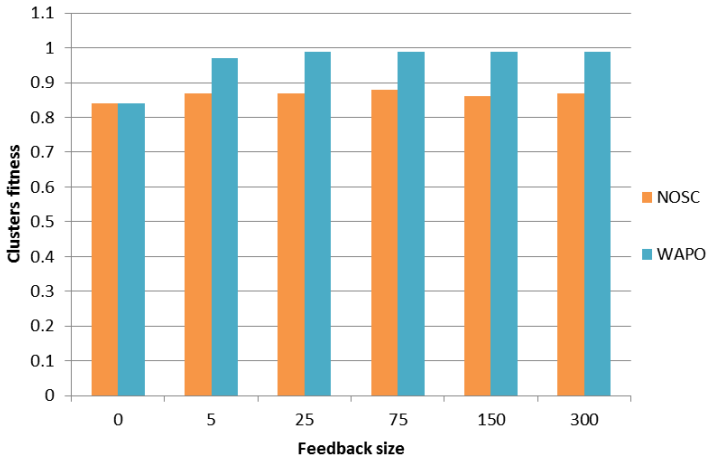
- We can see a positive result with WAPO which produced fitness scores from small amounts of feedback that are close to when larger amounts of feedback is supplied. This feedback represents just 0.1% from the total records, equalling to feedback level 5 in both AmazonGoogle and DblpAcm. The same positive result can be seen in AbtBuy where the 5 items of feedback represent 0.2% of its total records.
- However, WAPO produces a slightly less fit result at feedback amounts 25 and 75 in comparison with feedback level 5 (Figures 4.4a and 4.4b). This may seem counter intuitive but is actually not especially surprising. The reasons are:



(a) AbtBuy



(b) AmzonGoogle



(c) DblpAcm

Figure 4.4: Experiment 2 – Cluster fitness over different feedback sizes

- At all levels, the generated feedback represents a tiny subset of our ground truth. This means fitness scores may be susceptible to fluctuations as a result of the varying characteristics of the specific data that the feedback was drawn from.
- Individual runs of the evolutionary search cover specific, plausibly non-overlapping areas of the search landscape, obtaining unique candidates, which results in variations in result quality.

4.7 Discussion

From the evaluation, we learn that clustering alone does not necessarily produce a good integration. Some domain knowledge is needed. However, asking users to apply their knowledge on every record pair is infeasible for most real data sets; nevertheless, users may not find providing feedback troublesome if it involves only a small proportion of record pairs. We have learnt from our evaluation that simply applying user feedback through direct score changing does not produce satisfying results. A better approach is to *infer* knowledge from this small subset, and to apply this knowledge to the entire data set. Our evaluation shows this inference to be possible. We tested using a method of inference whereby a similarity measure is learnt in light of user feedback, and then used to cluster the rest of the data set. The result shows an improvement in the integration fitness against a chosen baseline. We extended our evaluation by adding parameters, used by the underlying clustering algorithm, to the search. The extension shows further improvement to the integration fitness, indicating the benefits obtainable from explicit user feedback. Our second evaluation investigated whether directly applying user feedback may cover a different integration landscape that cannot be identified by the optimiser. However, this is not observable with the selected datasets. This shows that optimisation with weights and configuration parameters generally clusters the data items for which there is feedback correctly, without the additional evidence that comes from changed scores.

4.8 Related Work

Many surveys on each step of instance integration have been published. Christen [Chr12] presents a survey on indexing/blocking techniques for instance resolution in large datasets. [KSS06, EIV07, DGdSM11, WLYF11] describe and compare

techniques to identify matching instances. Surveys on clustering instances for integration include [Rok10, HCLM09, XW05, JMF00]. Due to the presence of these extensive surveys, instead of producing yet another survey, in this section, we review state-of-the-art pay-as-you-go proposals related to our work by exploiting the following dimensions.

1. Pay method

Pay-as-you-go for instance integration delivers more than one interpretation notably to the method by which the *"payment"* is made. One view regards payment in the form of computational resource usage [WMGM13] whilst another view being the feedback from users about the correctness of one or more artefacts. This dimension describes the form of payment acceptable: *user feedback* or *compute resource*.

2. Touch points

PAYGO data integration has multiple stages where typically the use of user feedback could be found in one of these stages. Nevertheless, there are proposals that manipulate feedback in more than one stage. With more touch points along these lines of stages that can benefit from user feedback, this implies generally the extensiveness of feedback manipulation. Hence, this dimension bears the categories of *single* or *multiple*.

3. Method of inference

A user's knowledge of a domain is valuable but costly to collect. To apply this knowledge on every record pair is infeasible. Hence, the ability to infer similarity information based on a small subset of the data set that can be applied to the rest of the data set is a useful characteristic of an instance integration proposal. This dimension describes whatever method for inference is implemented by a proposal, if any.

Table 4.9: State-of-the-art pay-as-as-you-go proposals

Proposal	**Pay method**	**Touch points**	**Method of inference**
Corleone [GDD+14]	user feedback	multiple	trains a classifier using a small number of labelled data
CrowdEr [WKFF12]	user feedback	single	none
Whang *et al.* [WMGM13]	computer resource	multiple	none
Xiong *et al.* [XAF14]	user feedback	single	automatically forms new constraints based on constraints which have received vetting by user

We found four proposals of pay-as-you-go for integrating instances which manipulates domain information from user feedback.

Corleone [GDD+14] taps into a crowd to assist with integration decisions at multiple touch points. Upholding the notion of *hands-off crowdsourcing (HOC)*, Corleone lifts entirely any reliance on a *developer* to complete any task to integrate instances and hence promotes complete crowdsourcing. Stages which get crowdsourced in Corleone include *a*) blocking to reduce the number of pairwise comparisons, *b*) training of a learning-based matcher, and *c*) integration improvement for difficult cases. Corleone is interesting because it seizes user involvement as early as the blocking stage and at a greater extent than our proposal. While we propose an indirect user participation to blocking through the manipulation of a fitness function, Corleone directly includes users in the production of *"machine-readable blocking rules"* [GDD+14] by choosing rules that they perceive useful, and in the evaluation of the selected rules by assessing their precision. To infer similarity information, Corleone turns to the use of a classifier. The crowd's participation here is in labelling a small number of examples, fashioned from active learning, to train this classifier. Conversely, our proposal adopts an unsupervised learning approach, which works in the absence of a training set, and weights to infer similarity, delaying user's involvement until a cluster set has been obtained. For Corleone, improvement in the integration is supported by iterating through the stages with the addition of either making changes to an under-performing classifier or through the creation of a new specialised one.

CrowdEr [WKFF12] is another crowd-based proposal. Hinging upon the use of machine-based and user-based methods to identify matching entities, CrowdEr positions itself as a hybrid solution. CrowdEr consists of two primary phases. It starts by finding duplicates by way of machine, i.e., comparing text descriptions. Then, where *matches* exceed a decided threshold, they are presented to the crowd for clarity. In CrowdEr, matches that are to be validated by the crowd are formulated into Human Intelligence Tasks (HITs). A HIT is a "microtask" that a crowd participant is assigned to provide feedback. HITs come in two forms: pair-based and cluster-based. Multiple feedback is expected to be given with every cluster-based HIT, considering that it contains more than just a pair of records. Although most of the time the number of feedbacks equals the number of records in a HIT cluster, we view CrowdEr to be having a single touch point

as the feedback occurs in one stage. Since monetary payment is often involved in crowdsourcing, minimising the number of HITs is preferable, however, Wang *et al.* have shown that generating the minimum number of cluster-based HIT is an NP-hard problem, and thus sought the use of heuristics. We noticed that CrowdEr does not infer any similarity information from the set of feedback it receives. The assumption is most of the records have been resolved by the *"machine"* part of the proposal, and only a small number is left for the crowd. Hence, there is no need for any inference to be conducted. In contrast to our proposal, feedback is introduced as part of the integration process and not in some subsequent phase. Another difference is we do not specify which record pair needs clarifying. We allow a user to independently select the record pair to give feedback.

Whang *et al.* [WMGM13] is a pay-as-you-go proposal with *compute resource* as the payment type. In this proposal, *hints* are introduced which hold useful information for integrating instances. *Hints* can be either of the three forms: *sorted list of record pairs*, *hierarchy of partitions* or *sorted list of records*. Basically, hints are auxiliary data structures aimed at conducting record comparisons in the most profitable manner, defined here as *"with pairs that are most likely to match"* [WMGM13]. In other words, Whang *et al.* try to avoid comparing randomly ordered records because they can increase compute cost. Due to their heuristic nature, hints may or may not work with every entity resolution algorithm. There is an unavoidable tradeoff between the overhead of producing hints and the benefits obtained from using hints. No similarity information is inferred here.

Xiong *et al.* [XAF14] use a semi-supervised clustering approach to resolve instances while satisfying user-specified constraints. These constraints can be either a *must-link*, indicating two records are duplicates, or a *cannot-link* for non-duplicate record pairs. Xiong *et al.* perform iterative refinement to the current clustering model by dynamically consulting users for their knowledge. This exhibits a behaviour of a pay-as-you-go data integration. The key concern of this proposal is in the selection of informative record pairs worthy to take up the role of *"labelled examples"* in the cluster that they are to reside. Xiong *et al.* follow the intuition that improperly-selected labelled examples can impede the performance of a clustering effort. Probability and uncertainty are used to

calculate how informative a pair is. Xiong *et al.* adopt the concept of neighbourhoods. A neighbourhood is a special subset of a cluster. It is special because it contains record pairs that have received user vetting. We could view them to be representatives of the cluster whereby any decision for membership must first be evaluated against them. Once an informative record pair has been identified, users' knowledge (in the form of constraints) is sought to decide on which neighbourhood the pair should belong to. Calculation of similarity information takes place automatically between the pair and every existing member of the chosen neighbourhood after its inclusion. Comparing to Xiong *et al.*, feedback in our proposal is completely manual. This is because feedback in our proposal is given at the user's choice. Clustering membership in Xiong *et al.* depends entirely on users, whereas we opt for distance calculation as the determinant.

4.9 Summary and Conclusion

In this chapter, we have fulfilled the second part of our aim, i.e., *to investigate the use of pay-as-you-go approaches for instance level data integration, especially the identification of duplicates*, and accomplished our second objective of *devising a strategy to integrate instances from different large data sources, and that takes advantage of knowledge from users, in the form of feedback, about the domain of interest* (O2).

We answered RQ4 (*how can we devise a strategy to integrate instances from different large data sources in a dataspace*) by proposing a pay-as-you-go technique to integrate instances from different large data sources. This technique builds upon the hypotheses that instance integration is the problem of *incremental clustering* (RH4a); user's valuable knowledge of the domain can be useful to improve integration (RH4b); data sets are unique in their characteristics, requesting for parameters tuning of the underlying incremental clustering algorithm (RH4c); and due to the different needs from users on information, it is beneficial to have an objective function that is guided by user's domain understanding (RH4d). Our technique works on the concept of clustering to group similar records together in relation to their syntactic data value. Weights are applied, indicating the discriminative influence which an attribute has on the data set. In order to determine a good set of weights, we turn to evolutionary search and the use of user feedback to guide the search.

Two experiments were conducted as we sought to understand: firstly, is by

relying on just domain information elicited from user could yield good integration quality if integration is viewed as a black box (RQ5) and secondly, how effective is our strategies in integrating instances as the amount of user feedback varies (RQ6). The following describes our evaluation.

- The first empirical evaluation compares different variants of our proposed strategy against a baseline. All variants assume the underlying instance integration process as a black box. With regards to this, we explore optimisation of parameters of weights only, weights and an incremental clustering algorithm's configuration, and post optimisation score change. We also tested the case when no optimisation is conducted, but instead similar scores of record pairs are directly changed based on user feedback to give a point of comparison. We were interested to know the relative performance of these variants. From our results, we can see that optimisation of attribute weights generates fitter clusters than clustering without weights. However, directly changing similarity scores does not give better results. Nevertheless, optimising weights and parameters of the clustering algorithm together produces the fittest clusters when compared to the other variants.

- From the result of the first experiment, a second experiment was conducted on the best performing variant to understand the rate at which the quality of a clustering can be expected to improve as the amount of feedback collected grows. The results show that depending on the characteristics of the data set, there are differences in the fitness of the produced cluster. But fitness is better than the baseline across all data sets and in the cases we studied only a very small amount of feedback was needed to effect an improvement in the matching results.

We can conclude that using pay-as-you-go approach to integrate instances in a dataspace can produce fit clusters. This is achievable by putting together incremental clustering, tuning of parameters, and user-driven evolutionary search. Through this approach, it is possible for users to provide feedback only on a small amount of record pairs and has the integration taking affect onto the rest of the records in the dataspace. Nevertheless, scalability is a primary concern. This approach requires lengthy run time, making it less than practical. In the next chapter, we propose a pruning strategy to overcome this scalability issue.

Chapter 5

Scaling the Approach

A primary concern in our instance integration by pay-as-you-go approach is scalability. A prolonged time is needed to complete the run of our approach. This necessitates a more efficient strategy to be introduced. We investigated two forms of scalability strategies, *parallelism* and *pruning*, to handle efficiency. We use *parallelism* to simultaneously execute multiple instances of our clustering algorithm. Each instance receives as input a set of configuration parameters, randomly produced, initially, and subsequently, specially bred configuration parameters from carefully selected parents. On top of that, we performed *pruning* of our dataset and conducted a re-run of the integration to compare the fitness of clusters produced with and without pruned datasets.

In the first section of this chapter, we describe how we adopted parallelism into our work and which part of our proposal it was used for. We then continue to describe the platform that we employed, the HTC Condor (HTCondor)[1], in particular, its parallel-based application manager, the Directed Acyclic Graph Manager (DAGMan). The second section describes our pruning strategy which builds upon RH7a (*removing uninformative records to form a pruned data set used for integrating instances will not substantially degrade the quality of the integration*) and answers RQ7 (*how could a comparable integration quality be achieved by using only a fraction of a data set*) – while the third section presents two experiments that address RQ8 (*would we see a substantial improvement in the quality of integration if we use a pruned data set produced from our pruning strategy in place of a full data set*) and RQ9 (*are there any improvements in the time to complete when integration is conducted on our pruned data set in contrast to on a full data set*). This chapter ends with a summary and conclusion of our

[1] https://research.cs.wisc.edu/htcondor/

work.

5.1 Adopting parallelism

Existing works on matching of records involving parallelism include [KTR12, KR13, KL07, CZH+02]. These works distribute the task of comparing records for any similarity to different compute units of a parallel-computing platform to be run simultaneously. The primary aim is to reduce the total processing time and hence improve efficiency.

We use parallelism to implement our evolutionary search component. The idea behind parallelism is the simultaneous running of many independent processes on multiple available machines, fully utilising all resources an organisation has, in particular, the ones that tend to sit idly, for example, when office hours end. In our work, we aimed at running our *population* in parallel. As a recap, from our previous work in Chapter 4, we have adopted the use of a genetic algorithm to produce optimised configuration parameters and weight attributes; and after several tries, we have consequently decided upon a *generation* size of 70 and a *population* size of 50.

Depending on the *complexity* of the dataset, the time taken to execute a single run of the underlying clustering algorithm varies. Assuming an instance of the clustering algorithm requires, as an example, 10 minutes to run, a total clock time needed to run 50 individuals and 70 generations is

```
10 minutes x 50 individuals x 70 generation size = 35,000 minutes.
```

This equals to 24.3 days, which is too long. Therefore, by adopting parallelism, we estimate the time to now be

```
10 minutes x 70 generation size = 700 minutes,
```

equalling to just approximately 12 hours. This is more acceptable than when no parallelism is used. In our estimation, all 50 individuals would complete their runs at about the same time, which is equivalent to one individual.

Our proposal does not perform parallelism at the record comparison level nor at the blocking level. Since the longest single run of a clustering that we experienced with our chosen datasets is only up to 5 minutes, we do not see this as a necessity at the time of the research but would so in the future.

5.1.1 Parallelising using HTCondor

In the previous chapter, we have described our variants of pay-as-you-go instance integration. Here, we explain our employment of the HTCondor batch system to implement the optimisation component of our variants. HTCondor was developed at the University of Wisconsin-Madison (UW-Madison) by the Centre for High Throughput Computing. Its first installation, 15 years ago, was as a production system for the UW-Madison Computer Sciences department. Since then, HTCondor has evolved to finally become what it is today, a specialised batch system for managing compute-intensive jobs. Often, batch systems require the use of dedicated machines for job runs, however, HTCondor is designed to harness the resources of idle, non-dedicated machines in a specified pool. For our work, we have utilised a university-wide HTCondor run by the University of Manchester.

5.1.2 Directed Acyclic Graph Manager (DAGMan) Application

HTCondor offers several applications to manage parallel-running jobs. One that is a meta-scheduler for job execution is DAGMan. HTCondor has the primary responsibility of finding machines for program execution and yet does not perform any job scheduling, especially that which involves dependencies, hence, HTCondor relies on applications, like DAGMan, to submit programs to it in some user-preferred order. Besides bearing the task of a scheduler, DAGMan is also responsible for recovery and reporting on the submitted programs.

At its core, DAGMan performs a sequence of dependent tasks, whereby the input, output or execution of one job is dependent on the input, output or execution of another, and hence, a job would commence only after all its dependencies are satisfied. This sequence is represented in DAGMan as a directed acyclic graph (DAG), whereby the *nodes* identify the jobs, and the *edges* are the dependencies. Figure 5.1 shows a diamond DAG with four jobs represented by four nodes (i.e., A, B, C, D) and dependencies describing how jobs B and C would only execute after A has completed, while node D depends on the results of nodes B and C.

Using DAGMan, we too have a diamond-shaped graph with the middle level representing a collection of dynamically-populated nodes that runs independent instances of our clustering algorithm (refer Figure 5.2). In the beginning, the root node (i.e., node A) has the task of populating an initial set of individuals and at the end of each iteration the node is responsible for delivering a new set of configuration parameters and attribute weights that it receives from the breeder

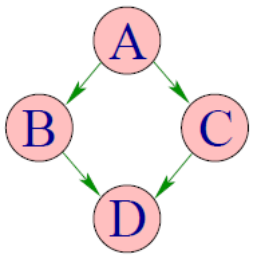


Figure 5.1: A directed acyclic graph.

node at the bottom of the loop, i.e., node C. There would be 50 instances of B node, representing the 50 individuals in the population. Leveraging DAGMan's *splice*, HTCondor iterates through the graph 70 times, and information is passed from one iteration to the next. The 70 times represent the number of generations

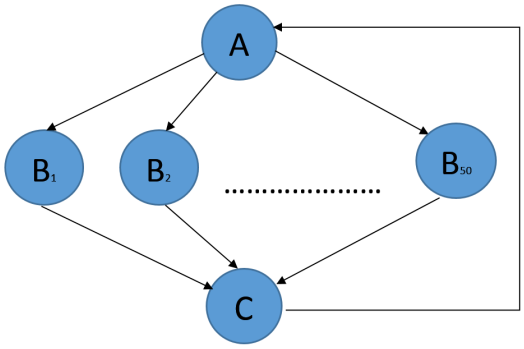


Figure 5.2: Our application of the directed acyclic graph.

5.2 Pruning the Search Space

Through parallelism we were able to improve on the total time taken to complete the integration. We echo the calculation here for *without parallelism used* and *when it is used*. *Without parallelism*, we obtained

```
10 minutes x 50 individuals x 70 generation size = 35,000 minutes,
```

which equals to 24.3 days. While *with parallelism*, it has improved to

```
10 minutes x 70 generation size = 700 minutes,
```

equalling to just 12 hours.

In practice, the amount of data continuously increases with time. We should then expect that our approach, even with parallelism, will eventually produce a

gradual uptrend of processing time. The reason is although our approach was able to produce fit clusters with the help from user feedback, however, the task of tuning the parameters requires numerous runs of the entire instance identification process. In conjunction with adopting parallelism, we propose a pruning strategy to reduce the search space. Fundamentally, pruning involves removing from a dataset records that, with some evidence, would not contribute to the resolution of entities. Hence, their removal poses no serious effects but instead can be beneficial to performance.

Another well-used scaling strategy that may not share the same spirit as *pruning*, but is not entirely unrelated, is *sampling*. Generally, *sampling* is the activity of carefully selecting items from a typically larger original collection, known as a *population*, with the purpose of representing the most number of unique items that can be found in that collection. In contrast, *pruning* typically engages in the removal of items, viewed to be useless, from a collection, leaving only useful ones. This difference is due to their different uses. *Sampling* can be found used in fully-supervised learning approaches [LC94, XL10] to form a static training set, and in active learning proposals [SB02, IAZ00, AED99, MMK00, SC00] to produce a more *interactive* training set. Uses of pruning have been found to include datasets [Hoy98], trees [MD97, MRA95, Nob02], and rules [Coh95, TKR$^+$95].

Placing the issue of scalability into our perspective, we propose a pruning strategy to mitigate costly executions of clustering over large datasets. Itemising the total cost that our proposal has to bear reveals the multiplication of the *population size*, the *number of generations* and the *cost of a single execution of clustering*. Many optimisation efforts use a model to estimate the fitness of candidate solutions in place of the actual task, like what can be found in a query optimiser, thus, avoiding the high cost that the running task could incur. However, in our case, obtaining a model would require us to be able to estimate, in advance, the effects of parameter changes. This is difficult to attain. As a result, we are left with a single alternative, that is to run the actual clustering and calculate the results for each candidate configuration. Such an approach has been adopted by others. For example, iTuned [DTB09] which is a tool that automates the tuning of database configuration parameters is triggered by the same motivation. In iTuned, different parameters responsible for its overall performance require the design of different experiments for their effects to be fully understood. Such a condition removes any possible deployment of an analytical cost model and

resorts to the running of the actual system. This is known as *closed-looped optimisation*, first used by Box [Box57]. Closed-looped optimisation is also utilised outside computing, ranging from analytical biochemistry performing instrument setup optimisation to improvements in chocolate production [Kno09].

We discuss in this chapter our proposal of a pruning technique aimed at reducing resource usage and response time. The application and evaluation of this technique fulfils our third objective (O3) — to design a technique which could integrate large amounts of data by using only a fraction of the complete dataset but without compromising the integration quality achieved when the approach in O2 is used.

5.2.1 Pruning our datasets

Although pruning may appear to be a straightforward task, it is not spared from inheriting complexity. Complexity can be found when deciding on what item to prune. Failure to identify relevant items may result in poor integration quality.

We define our pruned dataset as,

$$prunedDataset = \{r | r \in dataset, hasFeedback(r)\}$$

where r is a record in *dataset*, which refers to the original dataset, and r has been found to have been annotated. During optimisation, the pruned dataset is used in the same fashion as a full dataset, with the same dependence on syntactical distance of two records when calculating their degree of matching. However, the cluster fitness produced is based on just a subset of the complete dataset, and by clustering on the portion of the dataset which has obtained feedback, any weights and parameters produced represent estimates of the values that would be produced when used on the full dataset.

To reduce resource usage and response time, we devised a strategy which removes uninformative records. An informative record contains evidence that can be useful for the integration exercise. In our scenario, a distinction of an informative record is *association* with a user feedback. Generally, a user has knowledge about items in a particular domain, therefore, records with user feedback contain relevant information for use, establishing the suitability of configuration parameters, as they provide a subset of the ground truth. Building on this notion, our pruning technique leaves out records that are without feedback. Algorithm 5 provides its details in pseudo code.

Algorithm 5 Pruning strategy

```
1: allFeedback = get all user feedback
2: originalDataset = get all records of full dataset
3: for i ∈ originalDataset do
4:    if i ∈ allFeedback then
5:       append i into prunedDataset
6:    end if
7: end for
```

This special *association* with user feedback is parallel with our fitness measurement of clusters. We recall Section 4.3.3 which shows how our estimation of quality is derived from user's annotation on a subset of record pairs, marking what user perceive to be a match or a non-match.

Two concerns arise in connection with the formation of the pruned set. The first concern is the change in the collection of neighbours. This has the impact of altering the cluster membership of a record, which leads to the second concern — the ability to produce comparable quality to the original set. By having a smaller collection of data in order to speed up clustering, it offers a limited amount of information to find good configuration parameters. Eventually, a final cluster set is produced which is less fit than without pruning.

Pruning the original dataset involves assembling records that collectively are around twice the size of the given feedback at any point in time because a single item of feedback typically relates two records together. Where the number of records in the pruned dataset is less than twice the number of items of feedback, this is attributed to some records having more than one item of feedback. For example, records A and B are annotated as similar but record A is dissimilar with C and also receives an annotation. Thus, we have two items of feedback across three records instead of four. Table 5.1 displays the sizes of all pruned datasets. The smallest feedback is 5, while the largest is 300 which across all datasets is at most 25% of the total records in each dataset.

Table 5.1: Sizes of pruned dataset in light of different amounts of feedback

Dataset	**Total Number of Records**	**Feedback Amount**				
		5	**25**	**75**	**150**	**300**
AbtBuy	2173	10	50	146	28	535
AmazonGoogle	4589	10	50	149	295	578
DblpAcm	4910	10	50	150	292	572

For the very same reason as in earlier evaluations, every pruned dataset has the same amount of positive and negative feedback as the complete set except for feedback size 5 where the amount was randomly selected.

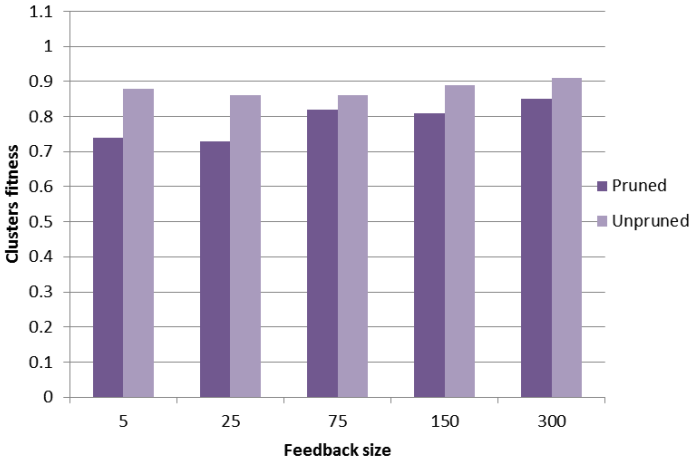
5.3 Experiments

The experiments presented in this section are in addition to the experiments conducted in Chapter 4. All aspects of the experimental setup from Chapter 4 is repeated here with the single difference of the data set used. In these experiments, our pruned data sets are used in place of the original full data sets. The underlying goal of conducting these experiments is to measure the degree of integration achievable through our proposed pruning strategy. These experiments help us to answer RQ8 (*would our pruning strategy be able to produce a higher degree of integration than when integration is done on the full data set*) and RQ9 (*would the time to complete an integration process be faster with a pruned data set produced based on hypothesis RH7a than when the complete data set is used*).

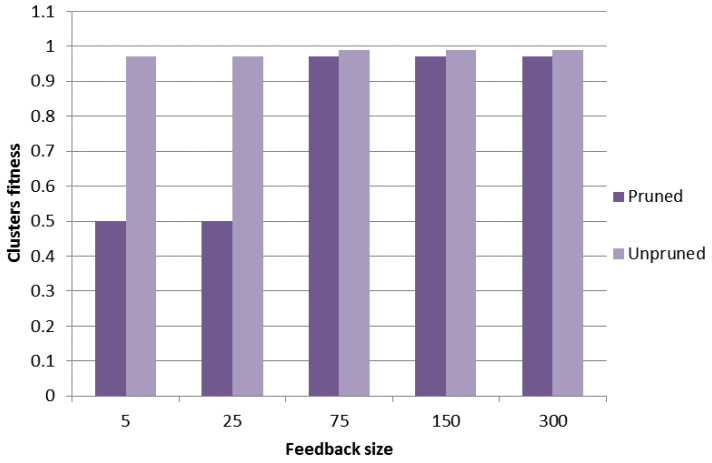
5.3.1 Experiment 3: Compare Quality of Optimised Clusters using Pruned Datasets

We conducted this experiment to understand the differences in the quality of clusters when pruned datasets are used with the optimiser instead of the complete dataset, where a pruned dataset almost always hosts a set of neighbours unlike that of the complete set. In ICA, neighbours participate in determining an instance's cluster membership. A cluster with the most neighbours similar to an instance will successfully include it. This is considered to be the *"preferred"* cluster for that instance (line 7 of Algorithm 4 in page 96). Intuitively, by having a large set of data, the pool of neighbours for an instance is more likely to be wider, giving the instance a better chance to find its *"preferred"* cluster. With a smaller neighbour pool, this perk may not always be available. Therefore, it is important to know if this change in neighbours will reduce the quality of the resulting clusters.

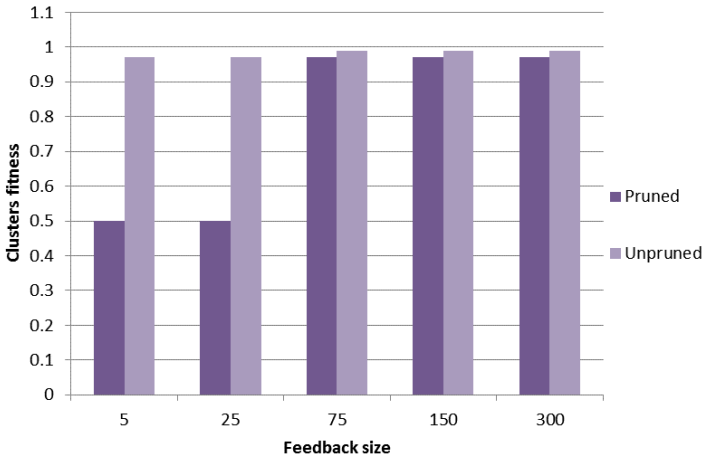
This experiment has the expectation to reach a comparable degree of quality with Experiments 1 and 2. The same set of feedback as these earlier experiments is used. Considering that WAPO has shown the most encouraging result with optimisation, this experiment has placed its focus on it. The following result can be observed.



(a) AbtBuy



(b) AmzonGoogle



(c) DblpAcm

Figure 5.3: Experiment 3 – Result

- In general, the result shows that optimising with the pruned dataset does not immediately provide quality equal to the complete dataset. However, quality improves as more feedback is received.
- In the beginning, the fitness markedly differs between the pruned and unpruned datasets, at feedback levels 5 and 25. DblpAcm shows the largest fitness difference, while the next largest difference is the pruned AmazonGoogle. AbtBuy with pruning fares slightly better than the other two datasets. This large gap demonstrates the insufficiency of information provided through the tiny amount of records of the pruned dataset.
- A turning point can be observed at feedback size 75, where an increase in fitness can be obtained with the pruned dataset across all cases. DblpAcm with pruning shows the largest improvement in fitness at feedback size 25. This is followed by pruned AmazonGoogle and pruned AbtBuy. These improvements are interesting because for AmazonGoogle and DblpAcm, the feedback amount of 75 represents only 2% of its full dataset, while for AbtBuy it represents 4%. Another interesting aspect is the increased fitness very nears the fitness received by the unpruned datasets. This jump in fitness indicates that the chosen feedback was able to provide the needed information to help the optimiser to search for better configuration parameters, leading to the improvement.

5.3.2 Experiment 4: Compare Clustering Times of Complete and Pruned Data sets

The purpose of this experiment is to ascertain the efficiency of the clustering run when the pruned dataset is used. We use WAPO in this experiment due to its effectiveness. Clustering of complete datasets practically consumes differing time range at each run. This difference can reach levels of up to around a factor of 3. Behind this phenomenon is the different collection of neighbours produced from the different sets of configurations continuously being churned from the optimiser. As such, we calculated the average *speedup* for each pruned dataset and define the total run time as a mean over 5 runs. We expect to gain faster clustering completion time than clustering with the full dataset.

Table 5.2: Average speed-up obtained using pruned dataset with 300 items of feedback

Dataset	***Speedup***
AbtBuy	27
AmazonGoogle	31
DblpAcm	63

From this experiment, we acquired the following results (see Table 5.2).

- Overall, all three datasets when pruned present *speedups* of several orders of magnitude compared with optimising with a complete dataset.
- The costly calculation of evaluating fitness of clusters can be substantially lowered by pruning, even when using large datasets.

5.4 Summary and Conclusion

Presented in this chapter is an integration technique that we have designed that *accepts a fraction of a data set and allows for integration across large data sources without compromising the integration's quality obtained from O2* (O3), hence answering RQ7. We have presented in this chapter, a technique that, based on user's feedback, selects records on which feedback has been provided. In other words, this technique prunes the complete data set by choosing only the records that have been validated by a user. We hypothesise that this manner of pruning will not compromise the quality of an earlier integration (RH7a). Since we have found that optimising weights and parameters can give fit clusters, we used that particular variant with our pruning technique.

Also presented in this chapter are two experiments. The first experiment answers RQ8 by comparing clusters generated through the pruning algorithm against those of the complete data set. Our evaluation shows that on most of the data sets, our proposed pruner produced comparably fit clusters at larger feedback amounts. The second experiment deals with the question of time to complete the integration process when the data sets are pruned (RQ9). From our evaluation, we found that the average run time with the pruned data sets is speedier by several orders or magnitude than with the complete data sets.

As a conclusion, pruning by removing uninformative records in a dataspace can produce fit clusters with more amounts of feedback. This deals with the scalability issue that was found in our instance integration technique described

in Chapter 4. Additionally, by pruning a data set, the run time performance also shows an improvement.

Chapter 6

Conclusions

In this chapter we review the contributions of our work and discuss future directions.

6.1 Review of Contributions

1. **An approach that ranks mappings based on their relevance to a user's information needs**

   We have contributed an approach to addressing the challenge of mapping proliferation attributed to the (semi-)automatic generation of mappings in dataspaces, by diagnosing it as a mapping ranking problem (RH1a). By tackling this proliferation, we obtained two benefits. Firstly, the search space during instance integration can be reduced, and secondly, we can expect to be presented with more accurate results. Our method tackled accuracy by identifying instances that are relevant to a user's information need (RQ1). These needs are interpreted from implicitly-provided feedback supplied by users, via the query logs. We held the view that a query's conditional clause provides terms that indicate a user's information need (RH1b). These terms can be used to rank semantic mappings (RH), whereby relevancy is calculated based on the rarity of terms found in a semantic mapping's extent (RH1d). Implicit feedback is not only abundant, but is also unobtrusive. This places it to be a very interesting candidate for further exploration. In this research effort, we have interpreted the conditional clauses of SQL queries that contain the *equality* operator. Other operators are left for future work. Our approach, to the best of our knowledge, is the first to address this challenge. We presented a ranking strategy which rests upon a widely-used

ranking scheme, the TF/IDF. In addition, we have suggested a variant to the TF/IDF to normalise the size of mappings, identified as the *size normalised TF/IDF (snTF/IDF)*.

2. **Empirical evaluations of the potential use of our approach in ranking mappings**

   We have designed experiments to know how different sizes of query logs affect the ranking score (RQ2), and to investigate how the rankings track query patterns that are skewed towards specific sources (RQ3). We view skewness to occur when some data sources contain a high frequency of terms that are not frequently found in the rest of the data sources (RH3a). Our experiments were not aimed at finding *correct* rankings, rather, we investigated properties of *stability* and *skewness*, because we took up the assumption that ground truth is innate to users, and hence we do not have easy access to such information. Both experiments yielded interesting results. For the first experiment, our approach was able to yield stable ranking even with rather small-sized query logs. The sizes are 400 when ranked using *snTF/IDF* and 500 for *TF/IDF*. In the second experiment, the rankings we produced exhibited sensitivity to the changes in skewness towards distinct data sources.

3. **A pay-as-you-go strategy to entity resolution introduced through disparate sources**

   To answer RQ4, we have designed and developed four variants of a pay-as-you-go strategy in resolving entities: No-optimisation score change (NOSC), Weight-only Optimisation (WOO), Weights-and-parameters Optimisation (WAPO) and Post-optimisation score change (POSC). These variants extended from a carefully chosen baseline and exhibited one or more of three interesting dimensions: first, adoption of optimisation, second, configuration parameters subjected to optimisation, and third, the direct changing of similarity scores by user feedback. Underlying our strategy are the following premises: i) instance integration in dataspace is the problem of incremental clustering (RH4a), ii) user possesses valuable domain knowledge which could improve integration (RH4b), iii) tuning the parameters of the underlying incremental clustering algorithm is necessitated by the unique characteristics of individual data sets (RH4c), and iv) to deal with the different information needs of users, a good approach is to use an objective function guided by

user's domain knowledge (RH4d). To implement optimisation, we used an evolutionary search, particularly, a genetic algorithm. By applying a genetic algorithm, the search effort receives guidance from a user on what are similar/dissimilar records. We modelled this guidance from a user as an objective function, which calculates the portion of records that comply with the user's perception. The obtained solution from the search and which had the greatest fitness than other solutions gets to form the final cluster set. To apply the genetic algorithm, we implemented operators that define the algorithm, i.e., *mutation*, *crossover* and *selection*. We have also designed the *phenotype* and *genotype*. We deployed our optimiser-based variant on a compute-grid environment, hinging upon *parallelism*. To understand the effectiveness of direct score changing, we used feedback from users to override any existing score produced from the underlying clustering algorithm. We identify an effective direct score-changing effort by how well it uncovers values located on the search space which the optimiser was not able to discover.

4. **Two evaluations which empirically examined the effectiveness of our instance integration proposal at two different levels of details**

   Two related experiments were designed to examine the effectiveness of our four variants when feedback is introduced. Through these experiments, RQ5 and RQ6 were answered. The first experiment aimed at identifying the most promising variant to be further evaluated in the second experiment. We opted for three levels of feedback for our first experiment, and use finer intervals of feedback levels in the second experiment to ascertain more closely the rate by which the quality of a clustering can be expected to improve with the growth in feedback. From the experiments, we found that user's domain knowledge plays an important role in the improvement of instance integration and combined with optimisation of algorithm-specific parameters produce fit clusters. Out of the four variants, WAPO was found to be the most promising, producing fit clusters even with a small amount of feedback.

5. **A pruning strategy that scales large data sets by removing uninformative records**

   We devised a pruning strategy that reduces the size of a large data set and yet able to maintain acceptable integration quality under certain conditions (RQ7). This strategy is aimed at reducing resource usage and response time, by emphasising informative records (RH7a). Such records have acquired

feedback from the user about their similarity and dissimilarity with one or more records. This is valuable information. Uninformative records are discarded with the confidence that they would not contribute to the quality of the produced cluster set. The reason is, in our setting, our pruning strategy reflects our evaluation formula, which in turn drives the evolutionary searchs objective function.

6. **An empirical investigation into the effectiveness and performance of our pruning strategy**

   We conducted two experiments aimed at comparing the level of effectiveness and efficiency achievable when pruned data sets substitute the complete data sets. Each experiment answers RQ8 and RQ9 respectively. Our description of this method can be found in Chapter 5. Pruned AbtBuy, AmazonGoogle and DblpAcm were used for these experiments, while the competing results of the complete data sets were taken from our earlier experiments. Additionally, from these experiments we re-used the same set of feedback to maintain constancy. Due to its encouraging results, we have employed WAPO for our pruning tests. The results show that the pruned data sets were not comparable in quality to their full counterparts when small amounts of feedback are supplied. Nevertheless, an improvement in the fitness can be seen as feedback number grows, starting from feedback size 75 for all pruned data sets. Impressively, this point of change represents from 2% to 4% of the full data sets. We could conclude that due to its dependence on user feedback, our pruning strategy is capable of producing a data set that, as more feedback is received through time, would assist the optimiser to produce a cluster set with a fitness that nears to a fitness obtained when a complete data set is used. With the pruned data sets requiring such a small percentage of the original data sets to trigger improvement in fitness (i.e., 2% to 4%), our test on the level of efficiency achievable with the use of pruned data sets reveals speedups of several orders of magnitude higher than when optimising with the complete data sets.

6.2 Future Directions

From our research, in this section, we describe open issues related to our proposals.

1. From the first part of our study, where we investigated the effect of varying

query log sizes to schema mappings ranking, we have learnt as more queries are issued by users, the more stable the ranking of mappings become. This shows implicit feedback to be a potentially reliable source of evidence for ranking schema mappings. However, we do need to take into account that the ranking is based on static result tuples from the mapping, which in reality is not always the case. One way is to capture a snapshot of the highly-ranked mappings' extent. Re-ranking can be conducted periodically to update changes.

2. Closely related to (1) is that we have defined three forms of mappings: *a*) *basic*, *b*) *refined* and *c*) *neutral*, which span two data sources. Over time, some mappings evolve through a dataspace's *refinement phase* and potentially affect a current ranking. In some cases, a basic mapping may transform into a refined mapping, and others may get deleted. Additional to the possible method proposed in (1), provenance information can be used prior to re-ranking. A recent work on provenance and mappings is Glavic *et al.* [GAMH10]. This work proposed two forms of provenance to firstly describe the association between transformed data and the relevant transformations, and secondly describe mappings that produced that data. Unlike both forms of proposed provenance, what we have in mind is more *mapping-centric*. We imagine provenance information should contain a history of expressions that a mapping previously assumed.

3. Provide meaning to other, more complex types of conditional expressions found in query logs besides *"equality"*. SQL is the instrument for users to specify information they wish to find, and it is also typically used for putting across specific information to be avoided. An example is the use of *set difference*, e.g., *NOT IN*. Another commonly-used SQL component is the *BETWEEN* operator. Future work could simply involve parsing an operator and populating it with relevant values. For example, the *BETWEEN* operator's value range would parse into a set of meaningful data items, which in our approach would emerge as a collection of terms to be discriminated.

4. Our experiments have been limited to relational databases. In reality, there are other forms of data model being actively used, for example object-oriented, XML, etc. Hence, it is only proper to expand our method to accommodate them.

5. Because of the direct dependency that our pruning strategy has on feedback

amount, the results we received from the use of pruned data sets have shown low fitness when feedback size is small. This fitness gradually improves as more feedback is supplied. However, in practice, the time needed to secure a good amount of feedback is not determinable. A possible approach is to adopt probabilistic methods to estimate the gains obtainable by which record and include that record into the pruned data set. Another candidate approach is to utilise available implicit feedback, which is plentiful and unobtrusive.

6. A particularly interesting work for the future is conducting data fusion for dataspaces. In this research, we have presented ways to find duplicate records from multiple disparate databases, but have not handled how these distinct versions of the same object are to be consolidated. A *pay-as-you-go feedback for data fusion* may implement the collection of strategies suggested by Bleiholder *et al.* [BN09]. However, it must consider the changeable nature of domains, in particular their data items, that have found the pay-as-you-go paradigm to be suitable. None of the strategies [BN09] has taken into account future updates to an already fused record pair. Such a characteristic is likely to warrant adjustments to the presented strategies.

7. Besides query logs, a host of other database artefacts can become implicit feedback. An especially useful implicit feedback for ranking of schema mappings is a user's *clickthroughs*. Often users would view a record which interests them, usually by clicking on either its hyperlink to expand viewing, by scrolling through the record; or downloading, printing, marking or tagging it. Statistics on clickthroughs could help infer relevant mappings.

8. Extend the depth of parallelism for our pay-as-you-go instance integration proposal to a finer granularity as can be seen done by [KTR12, KR13, KL07, CZH+02] — comparing records at each compute unit. A benefit expected is further improvement in efficiency.

Bibliography

[ABBG09] Paolo Atzeni, Luigi Bellomarini, Francesca Bugiotti, and Giorgio Gianforme. MISM: A Platform for Model-Independent Solutions to Model Management Problems. *Journal on Data Semantics XIV*, 5880:133–161, 2009.

[ABMM07] Yuan An, Alexander Borgida, Robert J. Miller, and John Mylopoulos. A Semantic Approach to Discovering Schema Mapping Expressions. In *23rd International Conference on Data Engineering*, pages 206–215. IEEE, 2007.

[ACD02] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: A System for Keyword-based Search over Relational Databases. In *18th International Conference on Data Engineering*, pages 5–16. IEEE, 2002.

[ACDG03] Sanjay Agrawal, Surajit Chaudhuri, Gautam Das, and Aristides Gionis. Automated Ranking of Database Query Results. In *1st Biennial Conference on Innovative Data Systems Research*. CIDR, 2003.

[ACM+08] Bogdan Alexe, Laura Chiticariu, Renée J. Miller, Daniel Pepper, and Wang-Chiew Tan. Muse: A System for Understanding and Designing Mappings. In *Proceedings of the International Conference on Management of Data*, pages 1281–1284. ACM, 2008.

[ACPS96] Sibel Adali, Kasim Seluk Cuk Candan, Yannis Papakonstantinou, and VS Siva S. Subrahmanian. Query Caching and Optimization in Distributed Mediator Systems. In *Proceedings of the International Conference on Management of Data*, volume 25, pages 137–146. ACM, June 1996.

[AED99] Shlomo Argamon-Engelson and Ido Dagan. Committee-based Sample Selection for Probabilistic Classifiers. *Journal of Artificial Intelligence Research*, 11:335–360, 1999.

[AEKV07] Rakesh Agrawal, Alexandre Evfimievski, Jerry Kiernan, and Raja Velu. Auditing Disclosure by Relevance Ranking. In *Proceedings of the International Conference on Management of Data*, pages 79–90. ACM, 2007.

[AGMS13] Yael Amsterdamer, Yael Grossman, Tova Milo, and Pierre Senellart. Crowd Mining. In *Proceedings of the International Conference on Management of Data*, pages 241–252. ACM, 2013.

[ANd06] Ajith Abraham, Nadia Nedjah, and Luiza de Macedo Mourelle. Evolutionary Computation: from Genetic Algorithms to Genetic Programming. In *Genetic Systems Programming: Theory and Experiences*, volume 13 of *Studies in Computational Intelligence*, pages 1–20. Springer, 2006.

[ASS09] Alsayed Algergawy, Eike Schallehn, and Gunter Saake. Improving XML Schema Matching Performance using Prüfer Sequences. *Data & Knowledge Engineering*, 68(8):728–747, 2009.

[BBBM06] Sugato Basu, Mikhail Bilenko, Arindam Benerjee, and Raymond J. Mooney. Probablistic Semi-supervised Clustering with Constraints. *Semi-supervised Learning*, pages 71–98, 2006.

[BC95] Shumeet Baluja and Rich Caruana. Removing the Genetics from the Standard Genetic Algorithm. In *Proceedings of the 12th International Conference on Machine Learning*, pages 38–46. Morgan Kaufmann Publishers Inc., 1995.

[BCC03] Rohan Baxter, Peter Christen, and Tim Churches. A Comparison of Fast Blocking Methods for Record Linkage. In *Conference on Knowledge Discovery and Data Mining*, pages 25–27. ACM, 2003.

[BDG+07] Lukas Blunschi, Jens-Peter Dittrich, Olivier René Girard, Shant Kirakos Karakashian, and Marcos Antonio Vaz Salles. A Dataspace Odyssey: The iMeMex Personal Dataspace Management System (Demo). In *3rd Biennial Conference on Innovative Data Systems Research*, pages 114–119. CIDR, 2007.

[BDPH06] Ladjel Bellatreche, Nguyen Xuan Dung, Guy Pierra, and Dehainsala Hondjack. Contribution of Ontology-based Data Modeling to Automatic Integration of Electronic Catalogues within Engineering Databases. *Computers in Industry*, 57(8):711–724, 2006.

[BGMK+06] Omar Benjelloun, Hector Garcia-Molina, Hideki Kawai, Tait Eliott Larson, David Menestrina, Qi Su, Sutthipong Thavisomboon, and Jennifer Widom. Generic Entity Resolution in the SERF Project. Technical report, Stanford InfoLab, 2006.

[BGMZ97] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic Clustering of the Web. *Computer Networks and ISDN Systems*, 29(8):1157–1166, 1997.

[BHN+02] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *18th International Conference on Data Engineering*, pages 431 –440. IEEE, 2002.

[BL04] Shawn Bowers and Bertram Ludäscher. An Ontology-driven Framework for Data Transformation in Scientific Workflows. In *Data Integration in the Life Sciences*, Lecture Notes in Computer Science, pages 1–16. Springer Berlin Heidelberg, 2004.

[BM03a] Mikhail Bilenko and Raymond J. Mooney. Adaptive Duplicate Detection using Learnable String Similarity Measures. In *Proceedings of the 19th International Conference on Knowledge Discovery and Data Mining*, pages 39–48. ACM, 2003.

[BM03b] Mikhail Bilenko and Raymond J. Mooney. On Evaluation and Training-set Construction for Duplicate Detection. In *Proceedings of the Knowledge Discovery and Data Mining Workshop on Data Cleaning, Record Linkage and Object Consolidation*, pages 7–12. ACM, 2003.

[BM07] Philip A. Bernstein and Sergey Melnik. Model Management 2.0: Manipulating Richer Mappings. In *Proceedings of the International Conference on Management of Data*, pages 1–12. ACM, 2007.

[BMCF03] M. Bilenko, R.J. Mooney, P. Cohen, and S.E. Fienberg. Adaptive Name Matching in Information Integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.

[BN09] Jens Bleiholder and Felix Naumann. Data Fusion. *ACM Computer Surveys*, 41(1):1:1–1:41, January 2009.

[Box57] George E. P. Box. Evolutionary Operation: A Method for Increasing Industrial Productivity. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 6(2):81–101, 1957.

[BPE+10] Khalid Belhajjame, Norman W. Paton, Suzanne M. Embury, Alvaro A. A. Fernandes, and Cornelia Hedeler. Feedback-based Annotation, Selection and Refinement of Schema Mappings for Dataspaces. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 573–584. ACM, 2010.

[BPF+11] Khalid Belhajjame, Norman W. Paton, Alvaro A. A. Fernandes, Cornelia Hedeler, and Suzanne M. Embury. User Feedback as a First Class Citizen in Information Integration Systems. In *5th Biennial Conference on Innovative Data Systems Research*, 2011.

[Bro02] Andrei Broder. A Taxonomy of Web Search. *Special Interest Group on Information Retrieval Forum*, 36:3–10, September 2002.

[Bro11] Jason Brownlee. *Clever Algorithms: Nature-Inspired Programming Recipes*. Jason Brownlee, 2011.

[CCMO11] Gianni Costa, Alfredo Cuzzocrea, Giuseppe Manco, and Riccardo Ortale. Data De-duplication: A Review. In *Learning Structure and Schemas from Documents*, pages 385–412. Springer, 2011.

[CDHW06] Surajit Chaudhuri, Gautam Das, Vagelis Hristidis, and Gerhard Weikum. Probabilistic Information Retrieval Approach for Ranking of Database Query Results. *ACM Transactions on Database Systems*, 31:1134–1168, September 2006.

[CFM+14] Valter Crescenzi, Alvaro A. A. Fernandes, Paolo Merialdo, Norman W. Paton, and Disheng Qiu. Crowdsourcing for Data Management: A Survey. 2014.

[Chr06] Peter Christen. A Comparison of Personal Name Matching: Techniques and Practical Issues. In *The 6th International Conference on Data Mining - Workshops*, pages 290–294, Dec 2006.

[Chr12] Peter Christen. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555, September 2012.

[CLM13] Matej Crepinsek, Shih-Hsi Liu, and Marjan Mernik. Exploration and Exploitation in Evolutionary Algorithms: A Survey. *ACM Computer Surveys*, 45(3):35, 2013.

[CMO10] Gianni Costa, Giuseppe Manco, and Riccardo Ortale. An Incremental Clustering Scheme for Data De-duplication. *Data Mining and Knowledge Discovery*, 20(1):152–187, 2010.

[Coh95] William W. Cohen. Fast Effective Rule Induction. In *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann Publishers Inc., 1995.

[Col11] Charles Cole. A Theory of Information Need for Information Retrieval that Connects Information to Knowledge. *Journal of the American Society for Information Science and Technology*, 62(7):1216–1231, 2011.

[Cor14] Paolo Cortez. *Modern Optimization with R.* Springer International Publishing, 2014.

[CQCS10] Huiping Cao, Yan Qi, K. Selçuk Candan, and Maria Luisa Sapino. Feedback-driven Result Ranking and Query Refinement for Exploring Semi-structured Data Collections. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 3–14. ACM, 2010.

[CR02] William W. Cohen and Jacob Richman. Learning to Match and Cluster Large High Dimensional Data Sets For Data Integration. In *Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining.* ACM, 2002.

[CSC04] Isabel F. Cruz, William Sunna, and Anjli Chaudhry. Semi-automatic Ontology Alignment for Geospatial Data Integration. In *Geographic Information Science*, pages 51–66. Springer, 2004.

[CSGM00] Junghoo Cho, Narayanan Shivakumar, and Hector Garcia-Molina.

Finding Replicated Web Collections. In *Proceedings of the International Conference on Management of Data*, pages 355–366. ACM, 2000.

[CSZ06] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, 2006.

[CT09] Xiaomeng Chang and Janis Terpenny. Ontology-based Data Integration and Decision Support for Product e-Design. *Robotics and Computer Integrated Manufacturing*, 25(6):863–870, 2009.

[CVDN09] Xiaoyong Chai, Ba-Quy Vuong, AnHai Doan, and Jeffrey F. Naughton. Efficiently Incorporating User Feedback into Information Extraction and Integration Programs. In *Proceedings of the 35th International Conference on Management of Data*, pages 87–100. ACM, 2009.

[CZH+02] Peter Christen, Justin Zhu, Markus Hegland, Stephen Roberts, Ole M. Nielsen, Tim Churches, and Kim Lim. High-performance Computing Techniques for Record Linkage. In *Australian Health Outcomes Conference*, 2002.

[DDCM13] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudr-Mauroux. Large-scale Linked Data Integration using Probabilistic Reasoning and Crowdsourcing. *The Very Large Database Journal*, 22(5):665–687, 2013.

[DGdSM11] Carina Dorneles, Rodrigo Gonalves, and Ronaldo dos Santos Mello. Approximate Data Instance Matching: A Survey. *Knowledge and Information Systems*, 27:1–21, 2011.

[DGL00] Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. Recursive Query Plans for Data Integration. *The Journal of Logic Programming*, 43(1):49–73, 2000.

[DHY09] Xin Dong, Alon Halevy, and Cong Yu. Data Integration with Uncertainty. *The Very Large Databases Journal*, 18:469–500, 2009.

[DKMR13] Susan B. Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Using the Crowd for Top-k and Group-by Queries. In *Proceedings of the 16th International Conference on Database Theory*, page 225. ACM Press, 2013.

[DR07] Hong-Hai Do and Erhard Rahm. Matching Large Schemas: Approaches and Evaluation. *Information Systems*, 32(6):857 – 885, 2007.

[DS06] Jens-Peter Dittrich and Marcos Antonio Vaz Salles. iDM: A Unified and Versatile Data Model for Personal Dataspace Management. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 367–378. VLDB Endowment, 2006.

[DSB09] Neelam Duhan, A. K. Sharma, and Komal K. Bhatia. Page Ranking Algorithms: A Survey. In *International Conference on Advance Computing*, pages 1530–1537. IEEE, March 2009.

[DTB09] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. Tuning Database Configuration Parameters with iTuned. In *Proceedings of the Very Large Databases Endowment*, volume 2, pages 1246–1257. VLDB Endowment, 2009.

[DU00] Jeffrey D. and Ullman. Information Integration using Logical Views. *Theoretical Computer Science*, 239(2):189 – 210, 2000.

[EEL11] Hazem Elmeleegy, Ahmed Elmagarmid, and Jaewoo Lee. Leveraging Query Logs for Schema Mapping Generation in U-MAP. In *Proceedings of the International Conference on Management of Data*, pages 121–132. ACM, 2011.

[EIV07] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19:1–16, 2007.

[ES98] Agoston E. Eiben and C. A. Schippers. On Evolutionary Exploration and Exploitation. *Fundamenta Informaticae*, 35(1-4):35–50, 1998.

[ES15] Agoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg, 2015.

[FDO+01] Dieter Fensel, Ying Ding, Borys Omelayenko, Ellen Schulten, Guy Botquin, Mike Brown, and Alan Flett. Product Data Integration in B2B e-Commerce. *IEEE Intelligent Systems*, (4):54–59, 2001.

[FHH+09] Ronald Fagin, Laura Haas, Mauricio Hernandez, Renee Miller, Lucian Popa, and Yannis Velegrakis. Clio: Schema Mapping Creation and Data Exchange. In *Conceptual Modeling: Foundations and Applications*. Springer Berlin Heidelberg, 2009.

[FHM05] Michael Franklin, Alon Halevy, and David Maier. From Databases to Dataspaces: A New Abstraction for Information Management. *ACM SIGMOD Record*, 34:27–33, December 2005.

[FKK+11] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. CrowdDB: Answering Queries with Crowdsourcing. In *Proceedings of the International Conference on Management of Data*, pages 61–72. ACM, 2011.

[FLM99] Marc Friedman, Alon Y. Levy, and Todd D. Millstein. Navigational Plans For Data Integration. In *American Association for Artificial Intelligence / Innovative Applications of Artificial Intelligence*, pages 67–73, 1999.

[FS69] Ivan P. Fellegi and Alan B. Sunter. A Theory for Record Linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, Dec 1969.

[FW97] March Friedman and Daniel S. Weld. Efficiently Executing Information-gathering Plans. In *Proceedings of the International Joint Conference of Artificial Intelligence*, 1997.

[GAMH10] Boris Glavic, Gustavo Alonso, Renée J Miller, and Laura M Haas. TRAMP: Understanding the Behavior of Schema Mappings through Provenance. In *Proceedings of the Very Large Data Bases Endowment*, number 1-2, pages 1314–1325, 2010.

[GC06] Karl Goiser and Peter Christen. Towards Automated Record Linkage. *Proceedings of the Fifth Australasian Conference on Data Mining and Analystics*, pages 23–31, 2006.

[GCBR05] Nizar Grira, Michel Crucianu, Nozha Boujemaa, and Inria Rocquencourt. Unsupervised and Semi-supervised Clustering : A Brief Survey. *A Review of Machine Learning Techniques for Processing Multimedia Content 1 (2005): 9-16.*, pages 1–12, 2005.

[GDD+14] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. Corleone: Hands-off Crowdsourcing for Entity Matching. In *Proceedings of the International Conference on Management of Data*, pages 601–612. ACM, 2014.

[GIJ+01] Luis Gravano, Panagiotis G. Ipeirotis, H. V. Jagadish, Nick Koudas, S. Muthukrishnan, and Divesh Srivastava. Approximate String Joins in a Database (Almost) for Free. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 491–500. Morgan Kaufmann Publishers Inc., 2001.

[GMPQ+97] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey Ullman, Vasilis Vassalos, and Jennifer Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[GMUW08] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, 2 edition, 2008.

[Gre86] John J. Grefenstette. Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 16(1):122–128, 1986.

[Gru93] Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[Gru01] Thomas Gruber. What is an Ontology? http://www-ksl.stanford.edu/kst/whatis-an-ontology.html, 2001. accessed on 01-09-2013.

[Hal01] Alon Y. Halevy. Answering Queries using Views: A Survey. *The Very Large Databases Journal*, 10:270294, 2001.

[HBF+09] Cornelia Hedeler, Khalid Belhajjame, Alvaro Fernandes, Suzanne Embury, and Norman Paton. Dimensions of Dataspaces. In *Dataspace: The Final Frontier*, volume 5588 of *Lecture Notes in Computer Science*, pages 55–66. Springer Berlin Heidelberg, 2009.

[HBM+12] Cornelia Hedeler, Khalid Belhajjame, Lu Mao, Chenjuan Guo, Ian Arundale, Bernadette Farias Lóscio, Norman W. Paton, Alvaro A. A. Fernandes, and Suzanne M. Embury. *DSToolkit: An Architecture for Flexible Dataspace Management*, pages 126–157. Springer Berlin Heidelberg, 2012.

[HBP+11] Cornelia Hedeler, Khalid Belhajjame, Norman W. Paton, Alvaro A.A. Fernandes, Suzanne M. Embury, Lu Mao, and Chenjuan Guo. Pay-as-you-go Mapping Selection in Dataspaces. In *Proceedings of the International Conference on Management of Data*, pages 1279–1282. ACM, 2011.

[HCLM09] Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Renée J. Miller. Framework for Evaluating Clustering Algorithms in Duplicate Detection. 2:1282–1293, August 2009.

[HdACC13] Carmem Satie Hara, Cristina Dutra de Aguiar Ciferri, and Ricardo Rodrigues Ciferri. Incremental Data Fusion Based on Provenance Information. In *In Search of Elegance in the Theory and Practice of Computation*, volume 8000 of *Lecture Notes in Computer Science*, pages 339–365. Springer, 2013.

[HFM06] Alon Y. Halevy, Michael Franklin, and David Maier. Principles of Dataspace Systems. In *Proceedings of the 25th Symposium on Principles of Database systems*, pages 1–9. ACM, 2006.

[HGP03] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient IR-style Keyword Search over Relational Databases. In *Proceedings of the 29th International Conference on Very Large Databases*, volume 29, pages 850–861. VLDB Endowment, 2003.

[HHH+05] Laura M. Haas, Mauricio A. Hernández, Howard Ho, Lucian Popa, and Mary Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *Proceedings of the International Conference on Management of Data*, pages 805–810. ACM, 2005.

[HIST03] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema Mediation in Peer Data Management Systems. In *Proceedings of the 19th International Conference on Data Engineering*, pages 505–516, 2003.

[Hoy98] Tetsuya Hoya. Graph Theoretic Techniques for Pruning Data and Their Applications. *IEEE Transactions on Signal Processing*, 46(9):2574–2579, 1998.

[HP02] Vagelis Hristidis and Yannis Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 670–681, 2002.

[HQC08] Wei Hu, Yuzhong Qu, and Gong Cheng. Matching Large Ontologies: A Divide-and-conquer Approach. *Data & Knowledge Engineering*, 67(1):140 – 160, 2008.

[HRO06] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data Integration: The Teenage Years. In *Proceedings of the 32nd International Conference on Very Large Databases*, pages 9–16. VLDB Endowment, 2006.

[HTMA13] Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Zoltán Miklós, and Karl Aberer. On Leveraging Crowdsourcing Techniques for Schema Matching Networks. In *Database Systems for Advanced Applications*, pages 139–154. Springer, 2013.

[IAZ00] Vijay S Iyengar, Chidanand Apte, and Tong Zhang. Active Learning using Adaptive Resampling. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining*, pages 91–98. ACM, 2000.

[IB13] Robert Isele and Christian Bizer. Active Learning of Expressive Linkage Rules using Genetic Programming. *Web Semantics: Science, Services and Agents on the World Wide Web*, 23:2–15, 2013.

[IFAB11] Tereza Iofciu, Peter Fankhauser, Fabian Abel, and Kerstin Bischoff. Identifying Users Across Social Tagging Systems. In *Proceedings of the 5th International Conference on Weblogs and Social Media*. Association for the Advancement of Artificial Intelligence, 2011.

[Ind01] Piotr Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithms*, 38(1):84–90, 2001.

[IRV13] Ekaterini Ioannou, Nataliya Rassadko, and Yannis Velegrakis. On Generating Benchmark Data for Entity Matching. *Journal on Data Semantics*, 2(1):37–56, 2013.

[Jai10] Anil K. Jain. Data Clustering: 50 Years Beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[JFH07] Shawn Jeffery, Michael Franklin, and Alon Y. Halevy. Soliciting User Feedback in a Dataspace System. Technical Report UCB/EECS-2007-38, EECS Department, University of California, Berkeley, Mar 2007.

[JFH08] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go User Feedback for Dataspace Systems. In *Proceedings of the International Conference on Management of Data*, pages 847–860. ACM, 2008.

[JGMP13] Manas Joglekar, Hector Garcia-Molina, and Aditya G. Parameswaran. Evaluating the Crowd with Confidence. In *Proceedings of the 19th International Conference on Knowledge Discovery and Data Mining*, pages 686–694. ACM, 2013.

[JMF00] Anil Kumar Jain, M. Narasimha Murty, and Patrick J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3):264–323, 2000.

[JSD+13] Shawn R. Jeffery, Liwen Sun, Matt DeLand, Nick Pendar, Rick Barber, and Andrew Galdi. Arnold: Declarative Crowd-Machine Data Integration. In *6th Biennial Conference on Innovative Data Systems Research*. CIDR, 2013.

[KHWL93] AH Kendrick, CM Higgs, MJ Whitfield, and G Laszlo. Accuracy of perception of severity of asthma: patients treated in general practice. *BMJ*, 307(6901):422–424, 1993.

[KL07] Hung-sik Kim and Dongwon Lee. Parallel Linkage. In *Proceedings of the 16th Conference on Information and Knowledge Management*, pages 283–292. ACM, 2007.

[KMS04] Nick Koudas, Amit Marathe, and Divesh Srivastava. Flexible String Matching against Large Databases in Practice. In *Proceedings of the*

13th International Conference on Very Large Databases, volume 30, pages 1078–1086. VLDB Endowment, 2004.

[Kno09] Joshua Knowles. Closed-loop Evolutionary Multiobjective Optimization. *Computational Intelligence Magazine, IEEE*, 4(3):77–91, 2009.

[Kon05] Grzegorz Kondrak. N-gram Similarity and Distance. In *Proceedings of the 12th International Conference on String Processing and Information Retrieval*, pages 115–126. Springer, 2005.

[KR13] Lars Kolb and Erhard Rahm. Parallel Entity Resolution with Dedoop. *Datenbank-Spektrum*, 13(1):23–32, 2013.

[KS91] W. Kim and J. Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *Computer*, 24(12):12 –18, dec 1991.

[KSS06] Nick Koudas, Sunita Sarawagi, and Divesh Srivastava. Record Linkage: Similarity Measures and Algorithms. In *Proceedings of the International Conference on Management of Data*, pages 802–803. ACM, 2006.

[KT03] Diane Kelly and Jaime Teevan. Implicit Feedback for Inferring User Preference: A Bibliography. *Special Interest Group on Information Retrieval Forum*, 37:18–28, September 2003.

[KTR10] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of Entity Resolution Approaches on Real-world Match Problems. In *Proceedings of the Very Large Databases Endowment*, volume 3, pages 484–493. VLDB Endowment, 2010.

[KTR12] Lars Kolb, Andreas Thor, and Erhard Rahm. Dedoop: Efficient Deduplication with Hadoop. In *Proceedings of the Very Large Databases Endowment*, volume 5, pages 1878–1881. VLDB Endowment, 2012.

[LC94] David D. Lewis and Jason Catlett. Heterogeneous Uncertainty Sampling for Supervised Learning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann Publishers Inc., 1994.

[Len02] Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *Proceedings of the 21th Symposium on Principles of Database Systems*, pages 233–246. ACM, 2002.

[Lev66] Vladimir I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, 1966.

[Lev98] Alon Y. Levy. The Information Manifold Approach to Data Integration. *IEEE Intelligent Systems*, 13:12–16, 1998.

[LLH11] Sanghoon Lee, Jongwuk Lee, and Seung-won Hwang. Scalable Entity Matching Computation with Materialization. In *Proceedings of the 20th International Conference on Information and Knowledge Management*, pages 2353–2356. ACM, 2011.

[Luk13] Sean Luke. *Essentials of Metaheuristics*. Lulu, 2013.

[MBR01] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic Schema Matching with Cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 49–58. Morgan Kaufmann Publishers Inc., 2001.

[MBWB99] D.G. Mayer, J.A. Belward, H. Widell, and K. Burrage. Survival of the FittestGenetic Algorithms Versus Evolution Strategies in the Optimization of Systems Models. *Agricultural Systems*, 60(2):113–122, 1999.

[MCD+07] Jayant Madhavan, Shirley Cohen, Xin Luna Dong, Alon Y. Halevy, Shawn R. Jeffery, David Ko, and Cong Yu. Web-Scale Data Integration: You Can only Afford to Pay as You Go. In *3rd Biennial Conference on Innovative Data Systems Research*, pages 342–350. CIDR, 2007.

[MD97] Dragos D. Margineantu and Thomas G. Dietterich. Pruning Adaptive Boosting. In *Proceedings of the 14th International Conference on Machine Learning*, volume 97, pages 211–218. Morgan Kaufmann Publishers Inc., 1997.

[ME+96] Alvaro E. Monge, Charles Elkan, et al. The Field Matching Problem: Algorithms and Applications. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 267–270. ACM, 1996.

[MG95] Brad L Miller and David E Goldberg. Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Complex Systems*, 9(3):193–212, 1995.

[MGMR02] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *Proceedings of the 18th International Conference on Data Engineering*, pages 117–128. IEEE, 2002.

[Mit14] Ruslan Mitkov. *Anaphora Resolution*. Routledge, 2014.

[MMK00] Ion Muslea, Steven Minton, and Craig A. Knoblock. Selective Sampling with Redundant Views. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence*, pages 621–626. AAAI Press, 2000.

[MNU00] Andrew McCallum, Kamal Nigam, and Lyle H Ungar. Efficient Clustering of High-dimensional Datasets with Application to Reference Matching. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining*, pages 169–178. ACM, 2000.

[MPE12] Ruhaila Maskat, Norman W. Paton, and Suzanne M. Embury. Pay-as-You-Go Ranking of Schema Mappings using Query Logs. In *Data Integration in the Life Sciences*, volume 7348 of *Lecture Notes in Computer Science*, pages 37–52. Springer Berlin Heidelberg, 2012.

[MRA95] Manish Mehta, Jorma Rissanen, and Rakesh Agrawal. MDL-Based Decision Tree Pruning. In *Proceedings of the 1st International Conference on Knowledge Discovery in Databases and Data Mining*, volume 21, pages 216–221. AAAI Press, 1995.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[MSD08] Robert McCann, Warren Shen, and AnHai Doan. Matching Schemas in Online Communities: A Web 2.0 Approach. In *24th International Conference on Data Engineering*, pages 110 –119. IEEE, april 2008.

[New67] Howard B Newcombe. Record Linking: The Design of Efficient Systems for Linking Records into Individual and Family Histories. *American Journal of Human Genetics*, 19(3 Pt 1):335, 1967.

[Nic97] David M. Nichols. Implicit Rating and Filtering. In *Proceedings of 5th Workshop on Filtering and Collaborative Filtering*, pages 31–36. ERCIM, 1997.

[Nob02] Andrew B Nobel. Analysis of a Complexity-based Pruning Scheme for Classification Trees. *IEEE Transactions on Information Theory*, 48(8):2362–2368, 2002.

[OK01] Douglas Oard and Jinmook Kim. Modeling Information Content Using Observable Behavior . *Proceedings of the Association for Information Science and Technology Annual Meeting*, 38:481, 2001.

[PB08] Rachel Pottinger and Philip A. Bernstein. Schema Merging and Mapping Creation for Relational Sources. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology*, pages 73–84. ACM, 2008.

[PF13] Norman W. Paton and Alvaro A.A. Fernandes. Crowdsourcing Feedback for Pay-as-you-go Data Integration. In *1st Very Large Databases Workshop on Databases and Crowdsourcing*, page 32. VLDB Endowment, 2013.

[PGMP+12] Aditya G. Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. Crowdscreen: Algorithms for Filtering Data with Humans. In *Proceedings of the International Conference on Management of Data*, pages 361–372. ACM, 2012.

[PI12] George Papadakis and Ekaterini Ioannou. Beyond 100 million entities: Large-scale blocking-based resolution for heterogeneous data. In *Proceedings of the 5th International Conference on Web Search and Data Mining*, 2012.

[PS05] Jaroslav Pokorny and Jozef Smizansky. Page Content Rank: An Approach to the Web Content Mining. In *Proceedings of International Conference on Applied Computing*, volume 1, pages 289–296. IADIS, 2005.

[RB01] Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The Very Large Databases Journal*, 10(4):334 – 350, 2001.

[RJ05] Filip Radlinski and Thorsten Joachims. Query Chains: Learning to Rank from Implicit Feedback. In *Proceedings of the 11th International Conference on Knowledge Discovery in Data Mining*, pages 239–248. ACM, 2005.

[Rok10] Lior Rokach. A Survey of Clustering Algorithms. In *Data Mining and Knowledge Discovery Handbook*, chapter 14, pages 269–298. Springer US, 2010.

[Rus18] Robert C Russell. Index, 1918. US Patent 1,261,167.

[SAR+07] Barry Smith, Michael Ashburner, Cornelius Rosse, Jonathan Bard, William Bug, Werner Ceusters, Louis J. Goldberg, Karen Eilbeck, Amelia Ireland, Christopher J. Mungall, et al. The OBO Foundry: Coordinated Evolution of Ontologies to Support Biomedical Data Integration. *Biotechnology*, 25(11):1251–1255, 2007.

[SB02] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive Deduplication using Active Learning. In *Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining*, pages 269–278. ACM, 2002.

[SC00] Greg Schohn and David Cohn. Less is More: Active Learning with Support Vector Machines. In *Proceedings of the 17th International Conference on Machine Learning*, pages 839–846. Morgan Kaufmann Publishers Inc., 2000.

[SCED89] J. David Schaffer, Richard A. Caruana, Larry J. Eshelman, and Rajarshi Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 51–60. Morgan Kaufmann Publishers Inc., 1989.

[SD06] Parag Singla and Pedro Domingos. Entity Resolution with Markov Logic. In *The 6th International Conference on Data Mining*, pages 572–582. IEEE, 2006.

[SE05] Pavel Shvaiko and Jrme Euzenat. A Survey of Schema-Based Matching Approaches. In *Journal on Data Semantics IV*, volume 3730 of *Lecture Notes in Computer Science*, pages 146–171. Springer Berlin-Heidelberg, 2005.

[SLB12] Joachim Selke, Christoph Lofi, and Wolf-Tilo Balke. Pushing the Boundaries of Crowd-enabled Databases with Query-driven Schema Expansion. In *Proceedings of the Very Large Databases Endowment*, volume 5, pages 538–549. VLDB Endowment, 2012.

[STZ05] Xuehua Shen, Bin Tan, and ChengXiang Zhai. Context-sensitive Information Retrieval using Implicit Feedback. In *Proceedings of the 28th Annual International Conference on Research and Development in Information Retrieval*, pages 43–50. ACM, 2005.

[TC07] Naiyana Tansalarak and Kajal T Claypool. QMatch–Using Paths to Match XML Schemas. *Data & Knowledge Engineering*, 60(2):260–282, 2007.

[TJM+08] Partha Pratim Talukdar, Marie Jacob, Muhammad Salman Mehmood, Koby Crammer, Zachary G. Ives, Fernando Pereira, and Sudipto Guha. Learning to create data-integrating queries. In *Proceedings of the Very Large Databases Endowment*, volume 1, pages 785–796. VLDB Endowment, August 2008.

[TKM01] Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning Object Identification Rules for Information Integration. *Information Systems*, 26(8):607–633, 2001.

[TKR+95] Hannu Toivonen, Mika Klemettinen, Pirjo Ronkainen, Kimmo Hätönen, and Heikki Mannila. Pruning and Grouping Discovered Association Rules. In *Machine Learning Workshops on Statistics, Machine Learning, and Discovery in Databases*. European Conference on Machine Learning, 1995.

[TLL+06] Jie Tang, Juanzi Li, Bangyong Liang, Xiaotong Huang, Yi Li, and Kehong Wang. Using Bayesian Decision for Ontology Mapping. *Web*

Semantics: Science, Services and Agents on the World Wide Web, 4(4):243–262, 2006.

[TMC12] Thanh Tran, Yongtao Ma, and Gong Cheng. Pay-less Entity Consolidation: Exploiting Entity Search User Feedbacks for Pay-as-you-go Entity Data Integration. In *Proceedings of the 3rd Annual Web Science Conference*, pages 317–325. ACM, 2012.

[Ull97] Jeffrey D Ullman. Information integration using logical views. In *Proceedings of the International Conference on Database Theory*, pages 19–40. Springer, 1997.

[WGM14] Steven Euijong Whang and Hector Garcia-Molina. Incremental Entity Resolution on Rules and Data. *The Very Large Databases Journal*, 23(1):77–102, 2014.

[WKFF12] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. CrowdER: Crowdsourcing Entity Resolution. In *Proceedings of the Very Large Databases Endowment*, volume 5, pages 1483–1494. VLDB Endowment, 2012.

[WLGM13] Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. Question Selection for Crowd Entity Resolution. In *Proceedings of the Very Large Databases Endowment*, volume 6, pages 349–360. VLDB Endowment, 2013.

[WLLH03] Yuk-Yin Wong, Kin-Hong Lee, Kwong-Sak Leung, and C.-W. Ho. A Novel Approach in Parameter Adaptation and Diversity Maintenance for Genetic Algorithms. *Soft Computing*, 7(8):506–515, 2003.

[WLYF11] Jiannan Wang, Guoliang Li, Jeffrey Xu Yu, and Jianhua Feng. Entity Matching: How Similar Is Similar. In *Proceedings of the Very Large Databases Endowment*, volume 4, pages 622–633, 2011.

[WMGM] Steven Euijong Whang, Julian McAuley, and Hector Garcia-Molina. Compare Me Maybe: Crowd Entity Resolution Interfaces. Technical report, Stanford University.

[WMGM13] Steven Euijong Whang, David Marmaros, and Hector Garcia-Molina. Pay-as-you-go Entity Resolution. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1111–1124, 2013.

[WMK+09] Steven Euijong Whang, David Menestrina, Georgia Koutrika, Martin Theobald, and Hector Garcia-Molina. Entity Resolution with Iterative Blocking. In *Proceedings of the 35th International Conference on Management of Data*, pages 219–232. ACM, 2009.

[XAF14] Sicheng Xiong, Javad Azimi, and Xiaoli Z. Fern. Active Learning of Constraints for Semi-supervised Clustering. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):43–54, 2014.

[XE04] Li Xu and David W. Embley. Combining the Best of Global-as-View and Local-as-View for Data Integration. In *The 3rd International Conference of Information Systems Technology and its Applications*, pages 123–136, 2004.

[XE06] Li Xu and David W. Embley. A Composite Approach to Automating Direct and Indirect Schema Mappings. *Journal of Information Systems*, 31:697–732, December 2006.

[XL10] Shasha Xie and Yang Liu. Improving Supervised Learning for Meeting Summarization using Sampling and Regression. *Computer Speech & Language*, 24(3):495 – 514, 2010.

[XW05] Rui Xu and Donald Wunsch. Survey of Clustering Algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–78, May 2005.

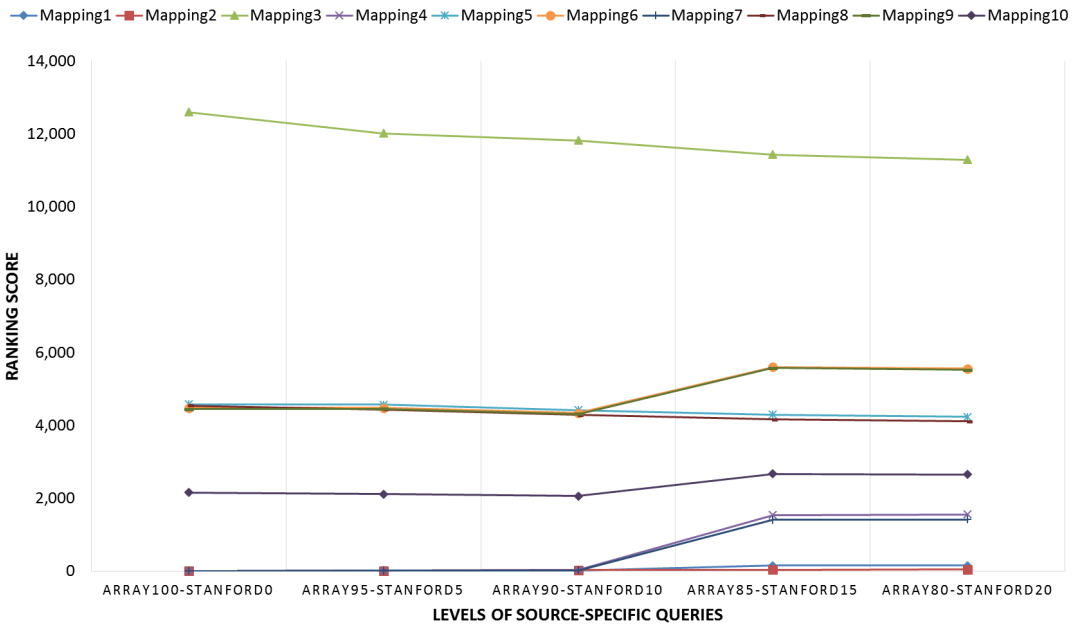
[YEE+10] Mohamed Yakout, Ahmed K Elmagarmid, Hazem Elmeleegy, Mourad Ouzzani, and Alan Qi. Behavior based Record Linkage. In *Proceedings of the Very Large Databases Endowment*, volume 3, pages 439–448. VLDB Endowment, 2010.

[ZCJC13] Chen Jason Zhang, Lei Chen, H.V. Jagadish, and Chen Caleb Cao. Reducing Uncertainty of Schema Matching via Crowdsourcing. In *Proceedings of the Very Large Databases Endowment*, volume 6, pages 757–768. VLDB Endowment, 2013.

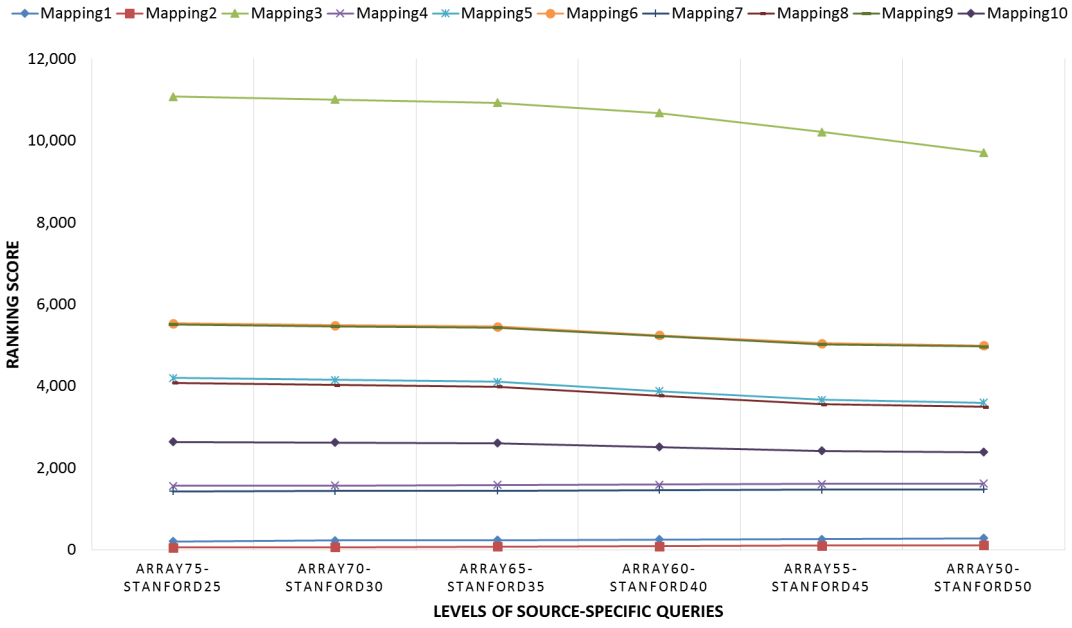
Appendices

Appendix A

Graphs of Ranking Result across Different Levels of Skewed Query Logs

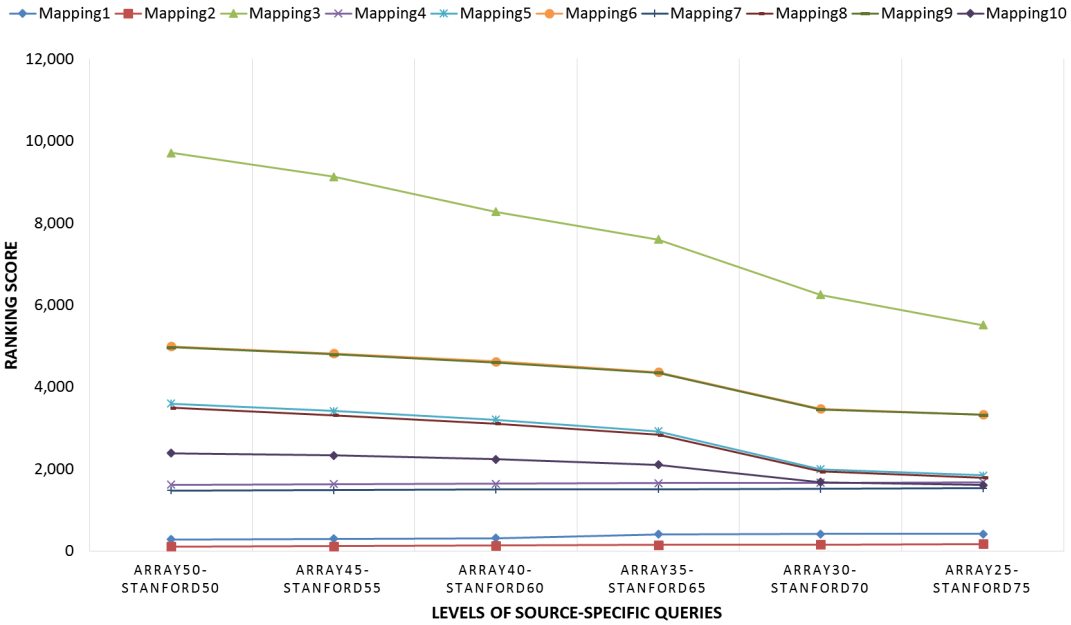


(a) Rankings: First quarter

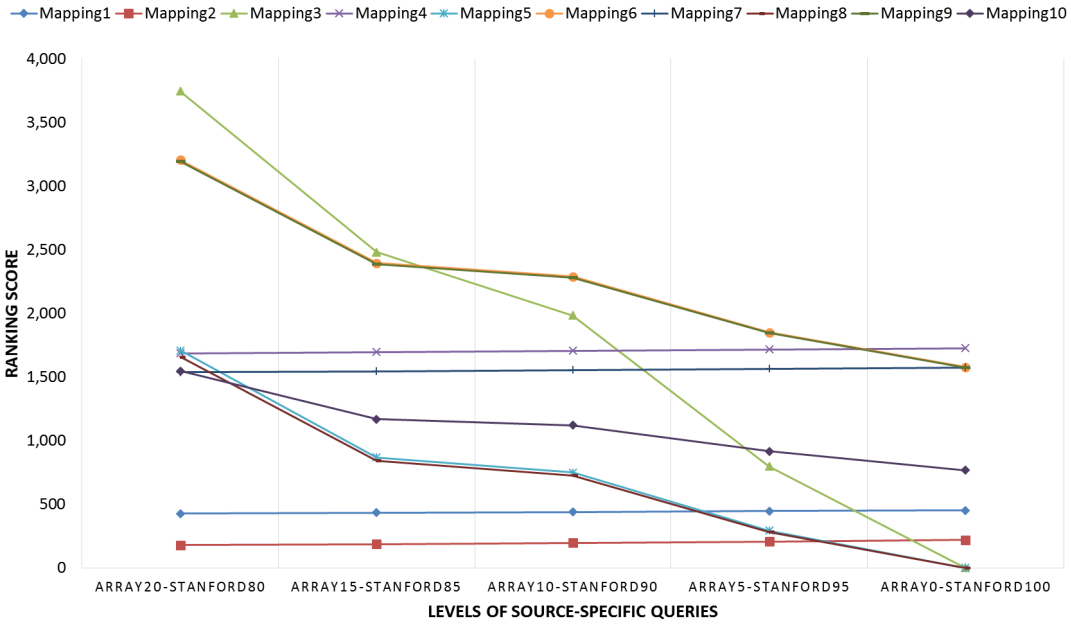


(b) Rankings: Second quarter

Figure A.1: TF-IDF ranking score for all mappings viewed across first and second quarters.



(a) Rankings: Third quarter

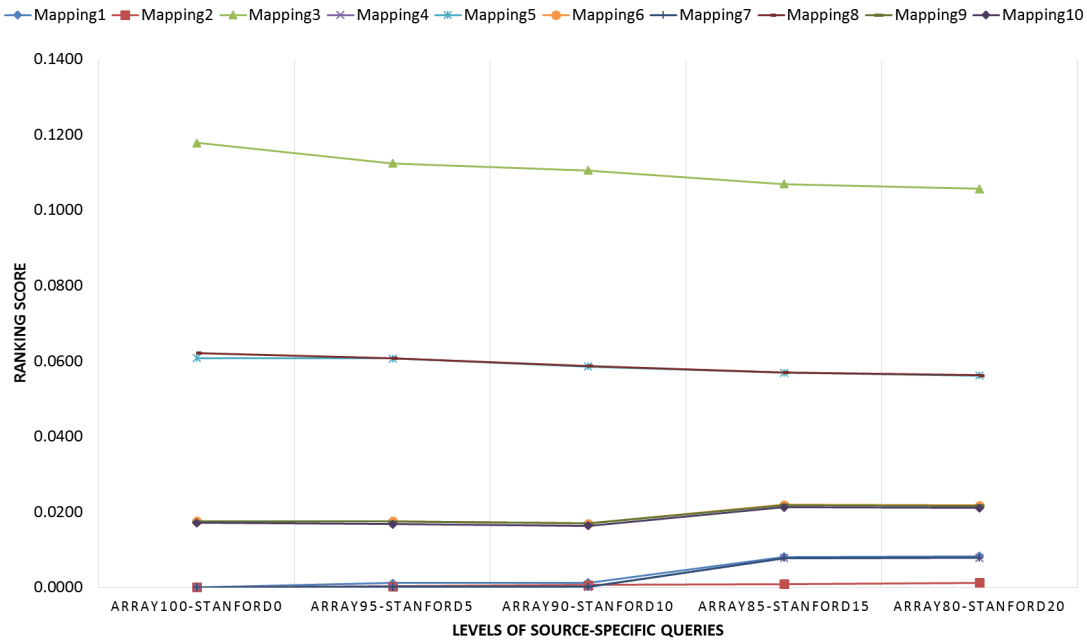


(b) Rankings: Fourth quarter

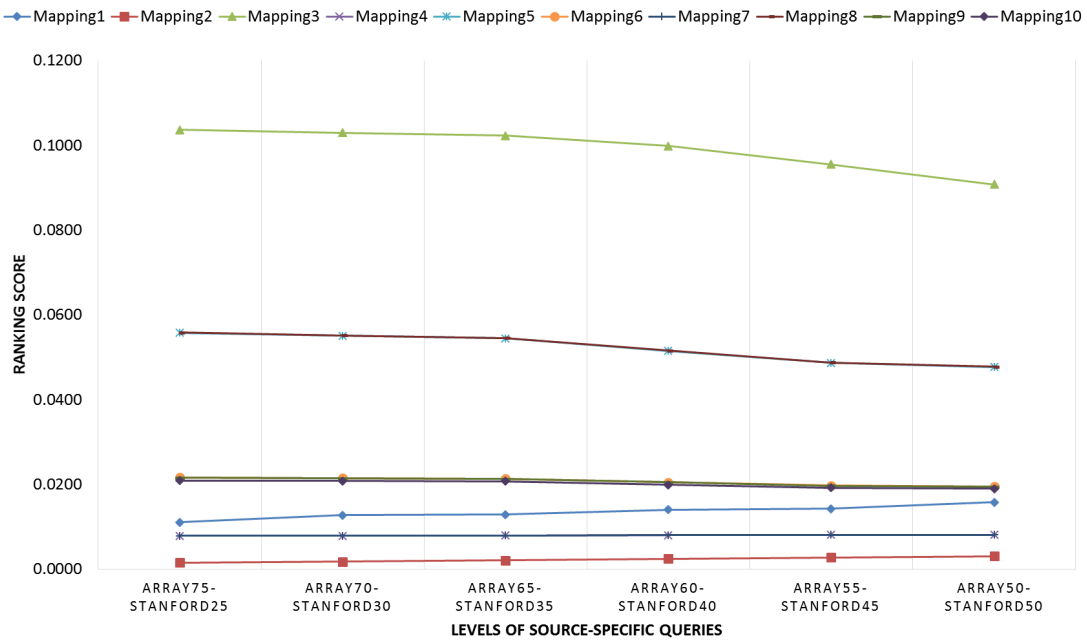
Figure A.2: TF-IDF ranking score for all mappings viewed across third and fourth quarters.

Appendix B

Graphs of Size Normalised Ranking Scores with Skewed Query Logs

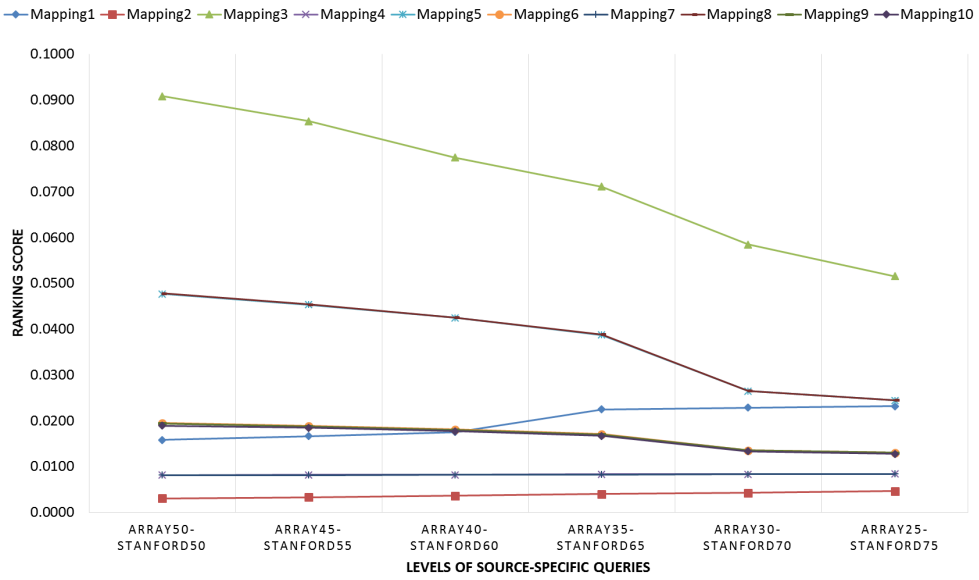


(a) Rankings: First quarter

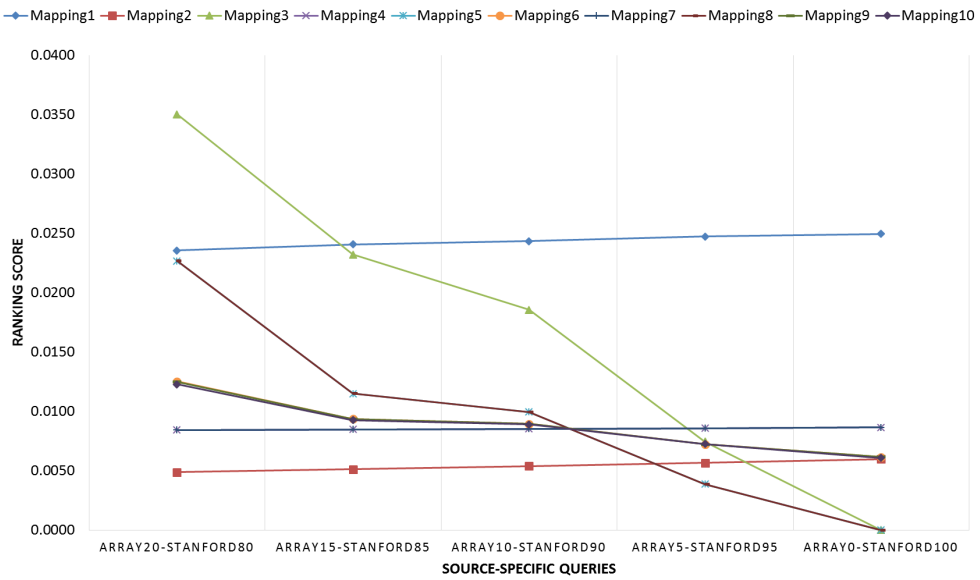


(b) Rankings: Second quarter

Figure B.1: Size Normalised TF-IDF ranking score for all mappings viewed across first and second quarters.



(a) Rankings: Third quarter



(b) Rankings: Fourth quarter

Figure B.2: Size Normalised TF-IDF ranking score for all mappings viewed across third and fourth quarters.

Appendix C

Process flow for Baseline of Pay-as-you-go Instance Integration

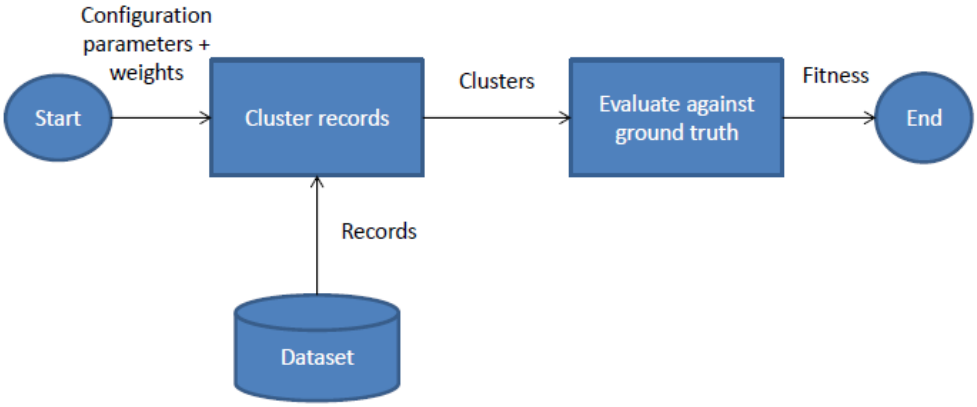


Figure C.1: Baseline – process flow.

Appendix D

Process flow for No-optimisation score change (NOSC)

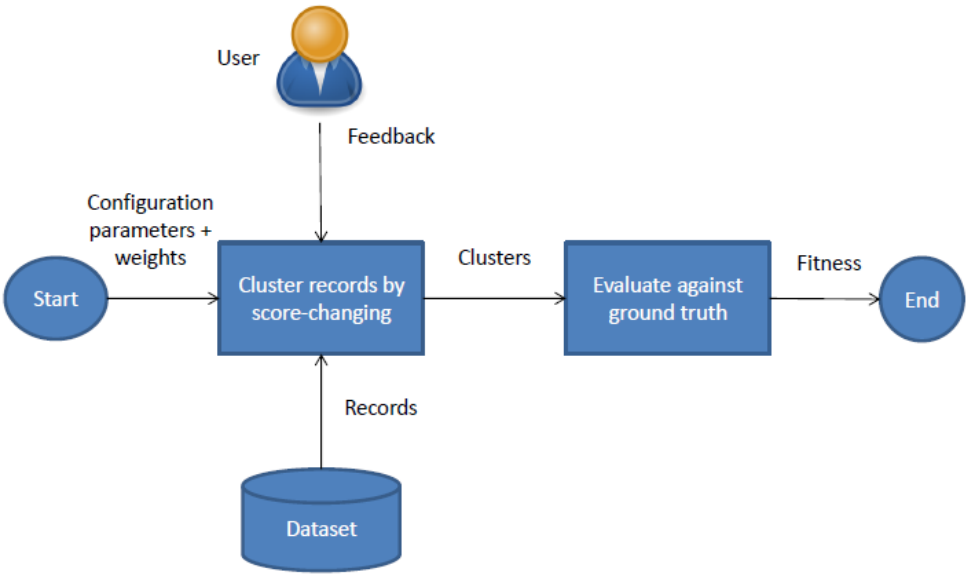


Figure D.1: No-optimisation score change (NOSC) – process flow.

Appendix E

Process flow for Weight-only Optimisation (WOO)

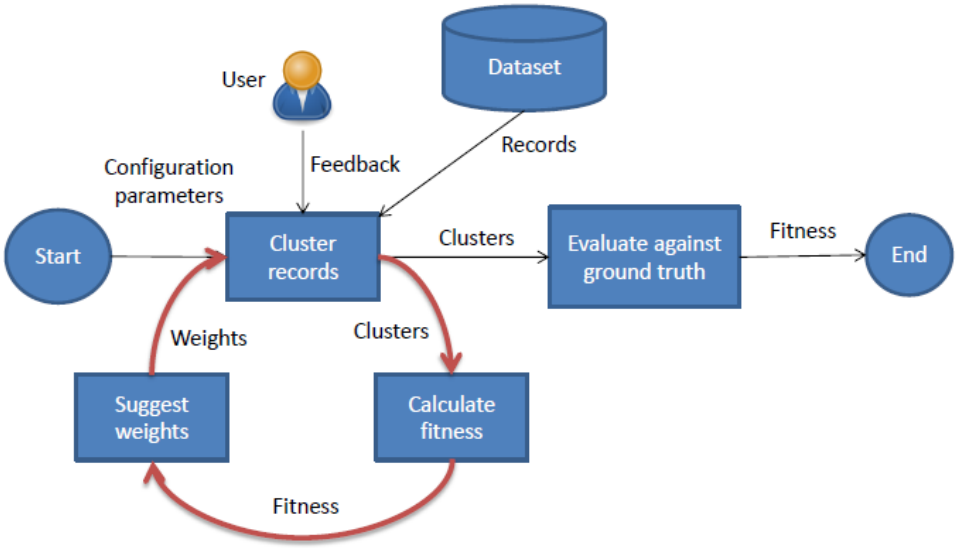


Figure E.1: Weights-only optimisation (WOO) – process flow.

Appendix F

Process flow for Weights-and-parameters optimisation (WAPO)

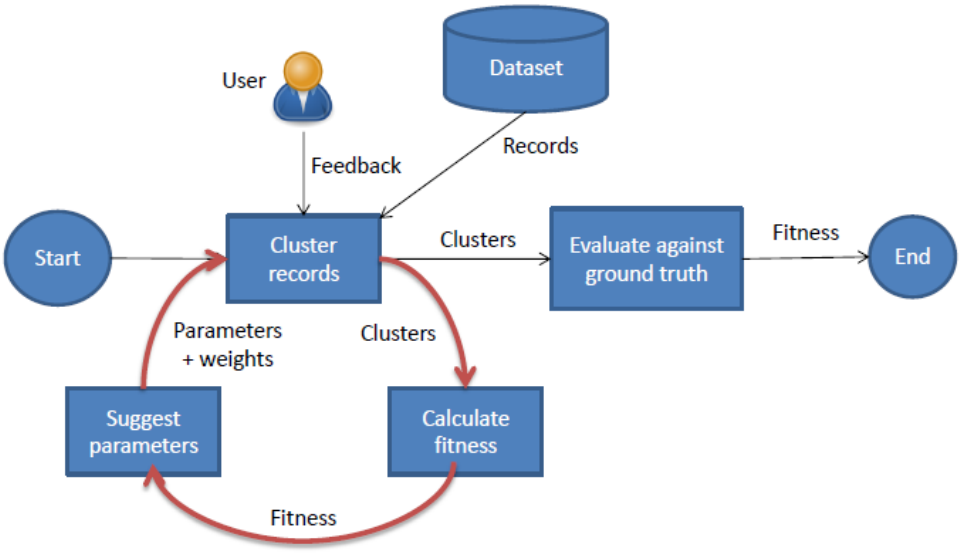


Figure F.1: Weights-and-parameters optimisation (WAPO) – process flow.

Appendix G

Process flow for Post-optimisation score change (POSC)

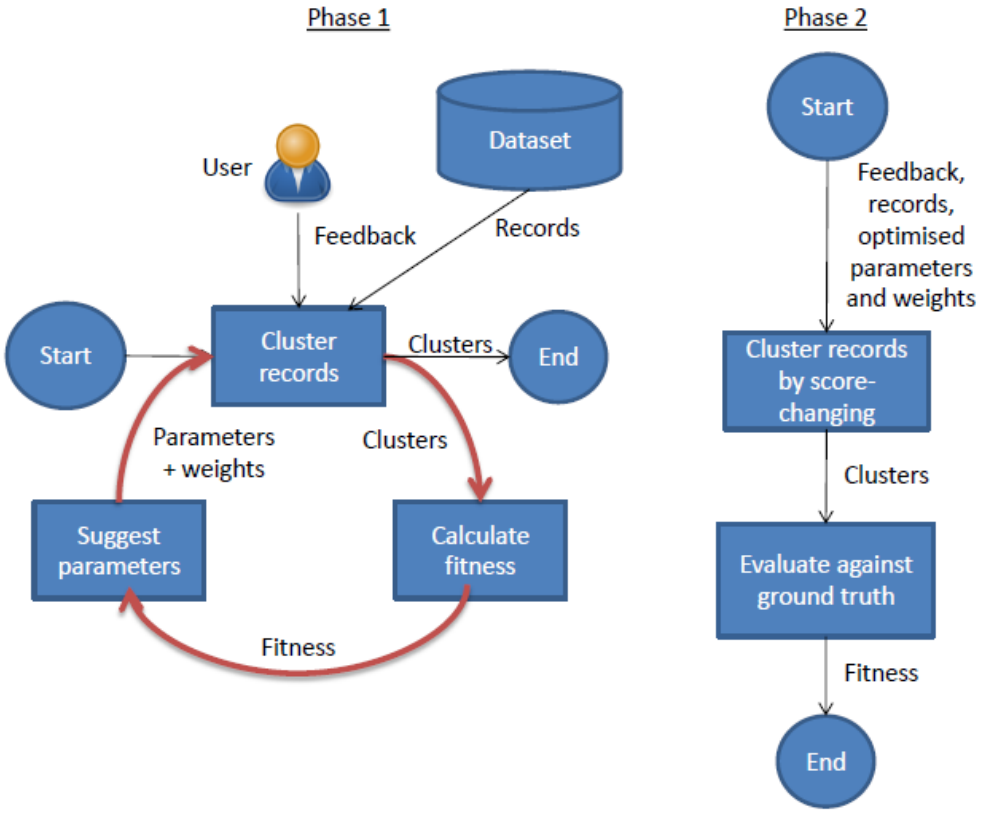


Figure G.1: Post-optimisation score change (POSC) – process flow.

Appendix H

Testing consistency in fitness across different runs

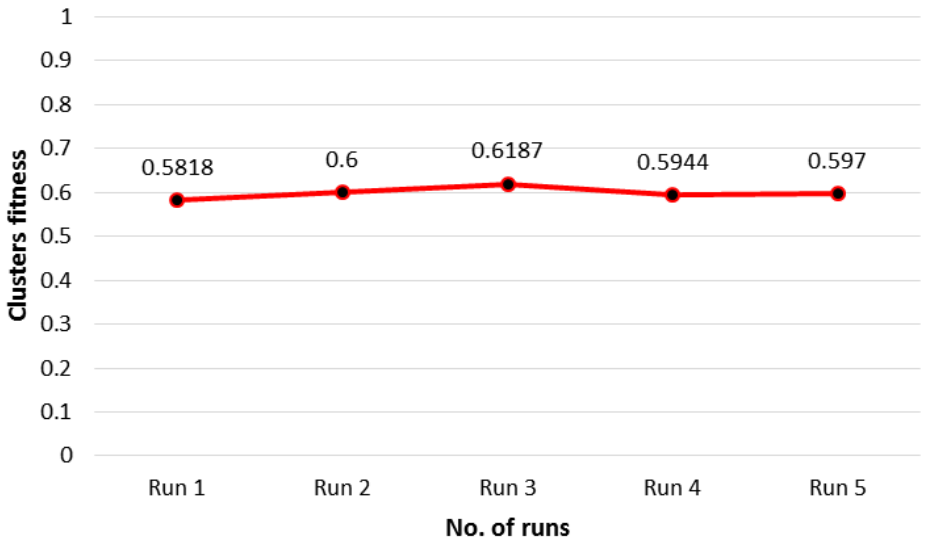
The aim of this set of runs is to observe the consistency in the fitness of the generated cluster set when different groups of random seeds are used to run the optimiser. We used WAPO for this test because of the following reasons:

- NOSC does not offer optimisation, hence it is not affected by random seeds. Furthermore, its underlying clustering algorithm is deterministic in light of the data set used and the sequence of new records introduced (this motivates the "incremental" characteristic of the clustering algorithm).
- WOO and POSC shares the same optimising component as WAPO, with the difference in the set of parameters to be optimised and the addition of directly changing similarity scores of record pairs. These differences, however, have no affect on the behaviour of the optimiser in the search of a good solution; WOO simply optimises a smaller subset of parameters than WAPO and the score change in POSC is executed after the entire run of the optimiser. We could consider POSC as WAPO followed by direct score changing. Hence, choosing any of the three optimiser-based variants would not make any difference in the consistency of the output.

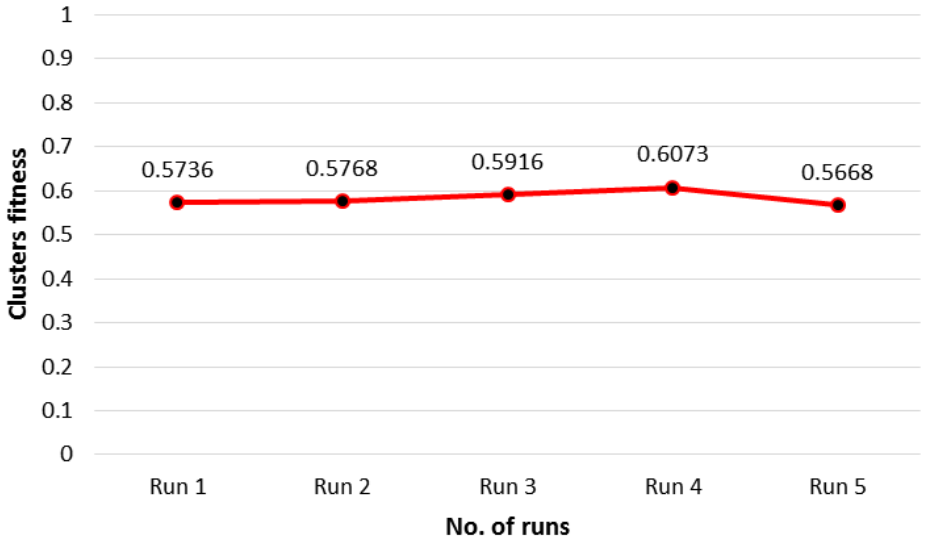
For these runs, we used only two of our chosen data sets, AbtBuy and Amazon-Google, reason being that both host complex data with much use of synonymous words and long, non-overlapping text. Five runs were conducted for each data set with a feedback amount of 250. This set of feedback was collected separately from the batches of feedback used in the experiments in Chapter 4 and thus may

guide the optimiser to a region different from what would be found in the experiments. The amount of 250 was chosen in order to have a size that has enough domain knowledge to make guiding of the optimiser fruitful.

The results (Figures H.1a and H.1b in page 166) show that across the five runs, for both data sets, there is consistency in the fitness produced. This implies that even with the use of random seeds the optimiser was able to learn from the provided user feedback and utilise the information to find good genotypes.



(a) AbtBuy



(b) AmzonGoogle

Figure H.1: Fitness consistency result