Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master's Thesis Nr. 93

Systems Group, Department of Computer Science, ETH Zurich

Joins based on the Access Path Model for Crowdsourced Databases

by

Lynn Aders

Supervised by

Prof. Dr. Donald Kossmann
Besmira Nushi

March 2013–September 2013

Informatik
Computer Science

Abstract

Integrating the crowd into established systems such as traditional relational databases poses new challenges to the design of a query optimizer. Not only the latency has to be considered, but also monetary costs and the accuracy of results. To achieve a better quality of the final answer, more money is needed to reward the workers. This problem apparently asks for trade-off solutions dependent on the users' needs and preferences. The notion of an access path as an independent source of information is the fundamental concept behind our considerations. The goal of the optimizer is to allocate the available budget to the given list of access paths in an ideal way.

This thesis presents a newly developed abstraction model for an optimizer joining data that is acquired at the same time over different information channels. The model consists of several parts such as an optimization model, an expectation model and a prediction model. The overall objective is to find the plan, i.e. the budget distribution among the access paths, which results in the highest expected utility. The utility is defined as the relative correctness of the query result.

The experimental evaluation shows the interplay of the different parameters related to the presented model. If one access path is clearly superior in comparison with the others in terms of cost and accuracy, then it will be favored by the optimizer. For balanced scenarios, it pays off to use a mixed plan which accesses several sources instead of only one.

Contents

Chapter 1

Introduction

This chapter motivates and introduces the multidimensional and therefore complex optimization problem of acquiring and joining data in a crowd-sourced setting. A summary of our contributions is given as an overview of how the problem was approached and what kind of potential applications will profit from our findings in the future. Finally, the content of each chapter is outlined to highlight the structure of this thesis.

1.1 Background and Motivation

Over the past decades, computers became increasingly more powerful and many new and more efficient algorithms in various fields such as *artificial intelligence* were developed to make use of the expanded computing capabilities. Nevertheless, there exist still many tasks in real-world scenarios where machines are limited in their skills and provide therefore incomplete or imprecise answers or in some cases even completely fail to execute a task.

Along with the evolution of new and better algorithms, many recent projects on the web use human-powered collaboration and participation of the members of their respective communities. The users usually contribute their knowledge or opinion in small parts to a large collection of information. The well-known crowdsourced online encyclopedia *Wikipedia* is a good example to demonstrate collective intelligence.

The idea of paying a monetary reward to users for completing such small tasks is the basic concept of crowdsourcing marketplaces like Amazon Mechanical Turk (AMT). These platforms enable requesters to issue and manage batches of microtasks and workers to find, accept and complete them. Examples for such microtasks include searching the web for missing information, joining related items together, labelling or segmenting images, transcribing audio files, comparing and rating items of a collection or even writing a summary of a text.

Out of this development evolved the intention to integrate the crowd's power into traditional relational database management systems (RDBMSs). Comparisons based on fuzzy criteria become feasible with the integration of the crowd, but the main goal was to overcome the drawbacks of the closed-world assumption in RDBMSs meaning that unavailable data is considered to be inexistent. The transition from a closed- to an open-world assumption poses new challenges to the design of an optimizer, because the optimization problem has more than one dimension in this scenario. The objective of acquiring new data from various sources of information and joining them together in a meaningful way are the driving forces behind our considerations.

1.2 Problem Statement

In a traditional RDBMS, the result of a query is strictly defined based on the current status of the database and therefore always the same, no matter how the declaratively stated query is executed internally. The optimization problem is therefore only considering the difference in execution time (or latency) needed for different possibilities to execute the query.

Integrating humans into the workflow of a complex system requires considering their reliability or in other words, their probability for returning correct and erroneous results respectively. Workers might submit wrong or irrelevant answers accidentally by having consulted the wrong source of information or in case they are not really qualified to complete the task to the requester's satisfaction. So-called *spammers* are simply interested in a maximum reward independent of the quality of the answers they are providing. To identify these answers and filter the results accordingly, a so-called *ground truth* must be known which is in general not the case. The question arises how the quality of answers is evaluated without knowing which items are relevant and which are not.

The assumption of the power behind crowd-enabled systems is that a large group of arbitrary individuals can outperform a smaller group of experts on the problem's domain simply by making use of the introduced redundancy. Thus the employed aggregation strategies and thresholds for workers' answers influence the performance of the crowd on the given task considerably.

Another important issue are the costs resulting from the monetary rewards paid to workers. In contrast to traditional relational database management systems where only the latency of a query operation needs to be considered, the optimization problem in the context of crowdsourced databases has therefore two more dimensions. The monetary costs paid to workers should be minimized while the quality of the answers should be maximized.

The degree of freedom in the crowdsourced setting refers to the various available information sources and it is therefore the optimizer's task to determine which origin of information is worth to be integrated in the final result to match the optimization objective. Thus trade-off solutions are required dependent on the user's needs and preferences, like for example what value of quality is achievable with a given budget.

1.3 Contribution

A new approach to join data originating from various information sources in the most optimal way is suggested in this work. Moreover, we are the first who consider joining data which is gathered at the same time from different sources in contrast to other approaches aiming at assembling and cleaning existing and previously known records. The goal is to integrate the power of the crowd which gets a monetary reward paid in return. Potential applications of our presented model in the future include first and foremost search engines, which are not yet able to aggregate, filter and join data from different sources of information (see example 1.1).

Example 1.1 (Movie screening in San Francisco) *Let us assume that we are interested in the movies screened in San Francisco around city centre next Saturday. Search engines will provide links to pages from different movie theatres that each in turn provide a list of the movies they are screening on individual days. Unless there exists a mashup site which integrates the different sources of information, each user who is interested in such a list of movies has to manually combine the different junks of knowledge.*

The formerly introduced concept of access paths in the crowdsourcing setting [5] denoting different sources of information is refined and adapted to the context of join queries with multiple relevant answers. Based on several assumptions, a *query optimizer* with an optimization model, an expectation model and a prediction model is suggested.

The *optimization model* makes use of a set of pre-defined access paths with properties related to their usage, reliability and identification. The notion of an access plan is used to characterize the number of questions acquired on each access path. The possibilities of aggregating workers' answers is described along with the resolution of ties.

The *expectation model* defines a score for each plan based on the expected utility specified as being the expected fraction of relevant items in the result set. Based on these calculated scores, the best plan which is still feasible for a given budget is selected.

The *prediction model* exhaustively enumerates all feasible plans for the given budget and for each plan all the vote sets to calculate the expected utility. The plan with the highest score is selected as the output of the optimizer and denotes how the system makes use of the available access paths in the presumably most efficient way.

Finally, simulated experiments are performed to explore the parameters' impact on the model. The sizes of the access paths' domains and the number of correct elements in the domains are investigated as well as various error rates.

1.4 Overview

This section provides an overview of the thesis and describes how the subsequent chapters and sections are organised.

The following chapter 2 deals with related work in the context of our problem statement. The first section, i.e. 2.1, highlights the notion of access paths introduced in traditional relational database management systems and how they are used in the optimizer to find cheap plans. Section 2.2 points out the popularity and various types of crowdsourced platforms with a strong focus on crowdsourcing internet marketplaces and crowdsourced databases. Section 2.3 describes the mathematical background for various considerations related to solving a system of equations and inequalities for our problem.

Chapter 3 introduces a model for an optimizer which can handle queries with multiple crowdsourced answers involved in a join operation. The notion of access paths is defined in our context as an abstraction to encode different information sources in section 3.1. The trade-off between quality, cost and latency is emphasized. Along with the parameters and abbreviations, the assumptions underlying our model are explained in great details. Then follow the decision-making functions in section 3.2, the resolution of ties in section 3.3 and a detailed description of the expectation and prediction models in the last two sections of this chapter.

The experimental results are explained in chapter 4. Simulation experiments highlight the different factors that influence the outcome of the score computation along with the magnitude of their impact.

The final chapter provides a summary of the results and contributions of this work. It also provides an outlook by listing potential tracks for future work.

Chapter 2

Related Work

Our work is related to many recent research topics in crowdsourcing and links also back to the beginning of traditional relational database management systems (RDBMS) by considering access paths as a means to retrieve data required in a declaratively stated query.

Crowdsourcing allows acquiring new or missing data, retrieving subjective comparisons and ratings and matching items based on fuzzy criteria. Crowdsourced databases integrate the power of crowdsourcing marketplaces into RDBMSs by automatically issuing microtasks when human-powered input is needed to return a satisfying answer. The paradigm shift from a closed-world to an open-world assumption poses new problems and significant challenges such as deciding whether an answer has a sufficiently large estimated quality or whether it is worth spending more money and time to hopefully improve the result.

Joins in the crowdsourcing setting can be categorized into two different classes. Joins on data which is available in the RDBMS can be seen as a special case of entity resolution. Many research works deal with topics related to this category. Another approach focuses on gathering new data while performing a join implicitly in the crowd.

The last section deals with various problems and their solutions related to integer linear programming. The goal is to use and adapt these techniques in the context of utility calculation in a crowdsourcing database optimizer. A system of linear equations and inequalities has to be stated first in order to apply these techniques.

2.1 Traditional Relational Databases

The first part of this section focuses on the concept of access paths and how they are used in an optimizer of a traditional relational database management system. The following subsection looks at the three most common implementations of joins in RDBMSs, namely *nested loops join*, *sort-merge join* and *hash join*.

2.1.1 Access Paths and Optimization in RDBMS

A traditional relational DBMS needs an optimizer in order to find one of the more efficient plans to retrieve the answer to a declaratively stated query. Efficiency in such systems is measured by the expected running time needed to retrieve the data according to that plan. Simple queries (involving a single relation) can have different access paths by using available indexes on that table or by doing a full table scan. More complex queries with several relations involved in joins or group-bys have in addition different orders of join execution which may vary greatly in performance [19].

The goal of the optimizer is to consider all combinations of access paths and to find approximately the cheapest plan, i.e. the one with the shortest execution time, to retrieve the data. The result set of tuples is equivalent for all plans, independent of the used access paths. It even pays off in most scenarios to use an expensive dynamic programming approach and collect fine-grained statistics on the tables, because the cost of different plans may vary in several orders of magnitude. Some plans are simply infeasible to retrieve the result, so an optimizer is absolutely needed in any such system [10]. Further optimization strategies were proposed by using approximations called *Iterative Dynamic Programming* for complex queries, where an exhaustive enumeration of plans is no longer feasible [11].

2.1.2 Implementations of Joins in RDBMSs

Most relational databases use three different implementations of joins as described in [15]. The results might be different in terms of tuple ordering and execution time, but not in terms of the set of tuples contained in it. As described in [10], these different implementations are considered in the *Method-Structure Space* of the query optimizer.

Nested-loops Join The nested-loops join is the straightforward implementation of the $\times - \sigma$ combination of relational algebra. Variants are the *Block Nested-loops Join* whose performance can be improved by using an in-memory hash table and the *Index Nested-loops Join* which uses an index on the inner relation to find matching tuples.

Sort-merge Join As a precondition, both inputs have to be sorted with respect to the join attribute(s). The merge join essentially merges both input tables as in sorting. It is typically used for equi-joins only.

Hash Join Partition both relations into n partitions using the same hash function h applied on the join attribute(s). By partitioning the data, the problem of joining is reduced to smaller sub-relations. Matching tuples are guaranteed to end up together in the same partition. By choosing n properly (i.e., the hash function h), partitions become small enough to implement the joins of sub-relations as in-memory joins. The in-memory join is typically accelerated using a hash table, too.

2.2 Crowdsourcing

Many popular platforms on the web that evolved over the past ten years use some sort of collaboration and participation of its users. Data is considered to be the driving force behind these rich, interactive, multimedia applications. Users contribute their knowledge or opinion in small junks to a large collection of information (i.e. collective intelligence). This new type of platforms is often referred to as Web 2.0 technologies. Examples include the famous online multilingual encyclopedia *Wikipedia*, the foto and video sharing platforms *Flickr* and *Youtube*, the blog creation and maintenance sites *Wordpress* and *Blogspot*, e-commerce platforms as *eBay* and *Amazon*, social media and community sites as *Facebook*, *Xing* and *Twitter*, and many more.

2.2.1 Crowdsourcing Internet Marketplaces

Crowdsourcing Internet Marketplaces provide access to a large community of people willing to perform small and usually simple tasks that a computer is not yet able to do. In return, these workers receive a small monetary reward as soon as the answers were accepted. Required qualifications help task requesters to ensure a certain quality on the results and to exclude poorly performing workers from their community. Tasks issued on such a platform could be one of the following: commenting on or linking to a blog entry, finding a phone number or mail address, labeling or rating images, writing or rewriting a paragraph or an article, translating a text, identifying actors in a movie, transcribing a podcast, and so on. *Amazon Mechanical Turk* (AMT) is the most famous and widely-used example for such a platform. Workers on AMT are usually called *Turkers* and the tasks are often referred to as *Human Intelligence Tasks (HITs)*.

Crowdflower is a service that simplifies access to the crowd by dividing larger tasks into smaller ones, providing appropriate HTML forms and processing the answers for their customers. The overhead of providing such templates and collecting the answers in a unified manner is outsourced to an expert in that field.

Several projects make use of the large community of workers available on such platforms and therefore the low latency of receiving an answer. For example a mobile application called *Vizwiz* that enables blind people to take a picture of their surroundings and get an associated question answered makes use of the crowd's abilities [2].

2.2.2 Crowdsourced Databases

Several research projects aim at integrating the power of crowdsourcing into traditional relational database management systems to enable the acquisition of new or unavailable data, to return subjective opinions and comparison results and to match items based on fuzzy criteria. A paradigm shift happens implicitly with this transition: From the closed-world assumption of an RDBMS, where only the data stored in relations is considered as being present and included in query answers, to an open-world assumption, where workers can give any answers and therefore generate new input. New crowd query operators need to consider the latency and also cleaning and aggregation of newly acquired data.

All published approaches consist of a programming layer on top of (possibly different) crowdsourcing platforms such as AMT. It provides a declarative interface usually based on an extension to SQL. Therefore, an optimizer is needed in all cases to reduce cost and latency while maximizing the quality.

Qurk Qurk is a declarative workflow query system that uses the underlying storage engine to retain results to previous queries for reuse. User-defined functions (UDFs) are used for aggregation of crowdsourced answers. Multiple responses to the same HIT from different workers are stored as a multi-valued attribute [14].

Deco Deco is based on so-called *conceptual relations* which are logical relations specified by the schema designer. To obtain the raw schema which is stored in a relational database, the conceptual relations' attributes are partitioned into *anchor attributes* for identification and *dependent attribute-groups* for specification of properties. *Fetch rules* modelled by the schema designer define how the data is gathered from the crowd by specifying a GUI and how much is paid for a satisfying answer given by a worker. *Resolution rules* are specified to aggregate workers' responses and at the same time remove inconsistencies in given answers [16, 17].

CrowdDB CrowdDB allows to crowdsource single attributes or whole tuples by providing various crowd operators such as *CrowdProbe*, *CrowdJoin* and *CrowdCompare*. Whenever more data is needed to produce a query result, CrowdDB consults the associated crowd, e.g. AMT, automatically and stores the obtained results in the underlying database [7]. In contrast to the other two approaches, the raw data is not stored in a database, but rather the cleaned and aggregated version obtained by a built-in majority-vote strategy [6].

Deco and Qurk allow a more fine-grained and user-defined aggregation and resolution of ambiguities compared to CrowdDB. Furthermore, it can change over time and results are adapted to new strategies since the raw data is stored in these two systems. In the crowdsourcing context, the optimization problem is different compared to the traditional relational model. Execution time should still be minimised as far as possible, but additionally and more fundamentally, monetary costs paid to crowd workers should be as low as possible. As a third goal, the quality of the answers is required to be maximized. Deco and Qurk give more weight to considering latency in the optimization and suggest therefore an asynchronous execution model.

Joins

Based on existing work related to joins, it turns out that some approaches perform a join on existing data while others try to gather new data during join execution. The first approach appears to result in the well-known *entity resolution* problem. The second one is related to *data gathering* with an implicit join performed in the crowd.

Entity Resolution Joining available data by comparing items based on fuzzy criteria is an entity resolution problem. The task of identifying matching items is performed by the crowd. Examples include matching images or video sequences from films with a table of celebrities and associating given categories to subjects of news articles in a collection. Various approaches have been suggested.

Marcus et al. [13] propose to use a (block) nested-loops join by using HIT comparisons to evaluate which elements match the join condition. This method results in a large number of tasks issued on the crowdsourcing platform, because the full cross product of tuples needs to be crowdsourced.

In the approach called *CrowdER* suggested by Wang et al. [22], a hybrid technique is applied. To avoid huge numbers of comparisons done by the crowd, an initial machine-based algorithm computes similarity measures.

The crowd is then used to verify whether the most uncertain pairs are matching and therefore referring to the same entity or not.

As a follow-up work [23], they looked at how to best invest the available budget in such a hybrid system by using transitive relations to lower the number of questions asked to the crowd. It turned out that it pays off to crowdsource first the potentially matching pairs and afterwards the non-matching pairs.

Whang et al. [24] use the power of the crowd to lead a probabilistic model towards a more accurate result. Each question is selected in such a way that its expected accuracy is maximum. Approximation algorithms are suggested, since calculating the expected accuracy turns out to be computationally infeasible.

Gathering New Data Acquiring new data in the crowd can be related to an implicit join operation performed by the workers. For example, if we ask for shops in Zurich and for each one what kind of products it sells. In that case, new tuples for shops and products are gathered while at the same time the matching relation is crowdsourced. Since the data tuples are not available at the beginning, this join operation is always performed as a nested-loops join. Hash or sort-merge join require the application of a hashing or sorting procedure, respectively.

Previous work by Trushkowsky et al. [20, 21] focused on estimating the number of elements in a collection by adapting species estimation techniques from biology. These estimators refine their answers while more questions are asked to the crowd and do not allow to determine a preliminary value before asking a single question. Obviously, this method is limited to finite and rather small sets. Unbounded or very large sets such as ice cream flavors do not allow to get accurate results in terms of the cardinality estimation.

Implementations of Joins in the Crowdsourcing Setting

Sort-merge join and hash join are applicable only if the data is already available, i.e. in the case of entity resolution. Additionally, an ordering or hashing function for the attributes of the join predicate is needed. If such a function is not defined for certain objects such as images or other media files, these special and often more efficient join implementations are not suitable. The more general nested-loops join can be used instead.

If new data is gathered while executing the join operation, it turns out that a nested-loops join is applied automatically. First, the tuples of the outer relation are crowdsourced and then for each of these the matching items for the inner relation are acquired. Therefore, a nested-loops join is applied implicitly and the other two join implementations are not an option in this case as described above.

Some sort of hash join called *Feature Filtering* is applied in Qurk [13] to reduce the number of comparisons on existing data. They group the tuples in each of the relations according to the features involved in the join predicate and then compare only the tuples in matching groups. This method reduces the search space significantly and has therefore in general a lower latency and lower costs.

2.3 Basis Computation and Integration over Lattices

The following subsections focus on mathematical tools and optimization strategies from linear algebra. They will be used later to check their usefulness in the context of an optimizer for crowdsourcing databases and integrate them as far as possible.

A system of linear equations and inequalities over nonnegative integers in n unknowns has a linear solution space, which is a subspace of $\mathbb{N}^n$. Various authors ([18], [8], [3]) focus their research work on finding a basis for this solution space, called a Hilbert basis. In general, this basis can be doubly exponential in size. It has been proven that integer linear programming, i.e. a linear optimization problem over integers, is $\mathcal{NP}$-complete [18, p. 245] and thus belongs to the hardest problems of the complexity class $\mathcal{NP}$.

The second subsection focuses on Barvinok's encoding [1] for lattice points inside a rational convex polyhedron. Barvinok found a polynomial algorithm for counting the number of such integral points inside a polyhedron specified by linear equations and inequalities in a fixed and rather small number of variables. For an increasing number of variables, the suggested approach becomes exponential again.

2.3.1 Basis of Solutions for a System of Linear Inequalities

The book *Theory of linear and integer programming* [18] summarizes important results related to linear programming (part 3) and integer linear programming ILP (part 4). Various problems and algorithms are considered along with their complexity compared to the size of the input. ILP is an optimization problem over the nonnegative integers or, equivalently, refers to searching for certain lattice points in a polyhedron specified by linear equations and inequalities.

Many research results and algorithms for polyhedra over rationals and reals do not hold in the setting of natural numbers or an equivalent formulation hasn't been found yet. In fact, integer linear programming was proven to be $\mathcal{NP}$-complete [18, p. 245] and therefore it is generally believed that these problems are not solvable in polynomial time. $\mathcal{NP}$-complete problems are the hardest problems in the complexity class $\mathcal{NP}$. Polynomial algorithms for ILP exist only for a fixed number of variables.

A Hilbert basis is a finite set of vectors which generates all integral points inside a polyhedral cone through nonnegative integral combinations of basis vectors. As summarized by Chubarov and Voronkov in [3], the cardinality of such a basis of the solution space (and therefore the complexity of computing and outputting it) is in the worst case doubly exponential in the number of variables. Such a computation is in most practical scenarios infeasible.

Hemmecke [8] suggests an algorithm to compute a minimal generating set of vectors for all nonnegative integral points inside a given lattice and in particular computing the Hilbert bases of cones.

2.3.2 Counting Lattice Points in a Convex Polyhedron

Counting the number of lattice points in a convex polyhedron specified by linear equations and inequalities for a fixed number of dimensions was revolutionized by Barvinok in 1994 [1]. Until then, polynomial algorithms were only known for at most 4 dimensions, i.e. $d \leq 4$.

Barvinok's encoding [4] considers lattice points as monomials and encodes them in a generating function as a sum of these monomials as follows:

$$f(K) = \sum_{\alpha \in K \cap \mathbb{Z}^d} z_1^{\alpha 1} z_2^{\alpha 2} \dots z_d^{\alpha d}. \tag{2.1}$$

This sum might become a very large or even infinite formal power series and therefore Barvinok suggested a compact encoding as a sum of rational functions which is of polynomial size and can be determined in polynomial time in the size of the input for a fixed dimension d:

$$f(K) = \sum_{i \in I} E_i \frac{z^{u_i}}{\prod_{j=1}^{d} (1 - z^{v_{ij}})} \tag{2.2}$$

where I is a polynomial-size indexing set, $E_i \in \{1, -1\}$ and $u_i, v_{ij} \in \mathbb{Z}^d \; \forall i, j$. In case d is not fixed, counting the number of lattice points inside a convex polyhedron becomes exponential again.

Example 2.1 *Consider a triangle as shown in figure 2.1. The vertices are given as follows: $V_0 = (0,0)$, $V_1 = (6,0)$ and $V_2 = (2,2)$. For each vertex of the polyhedron*

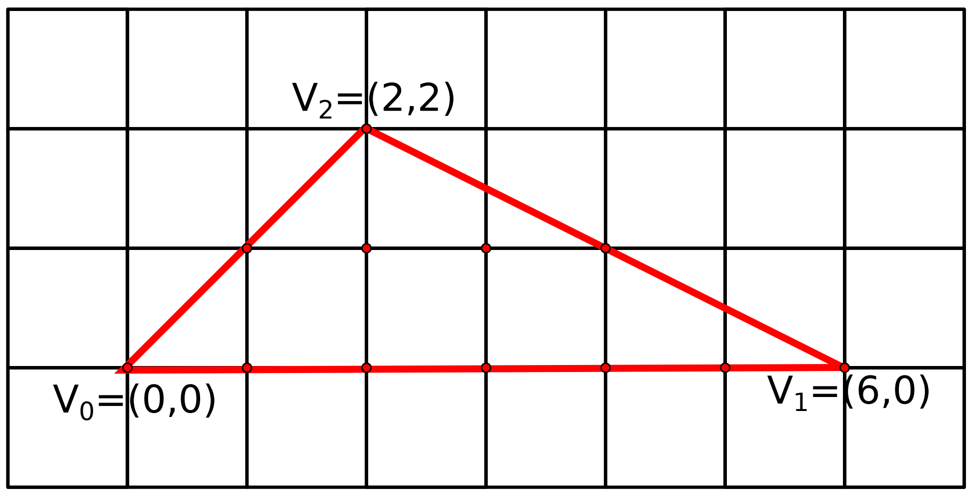


Figure 2.1: The triangle explained in Example 2.1.

a rational generation function is obtained as described by Barvinok:

$$f(K_{V_0}) = \frac{1}{(1-z_1)(1-z_1z_2)}$$
$$f(K_{V_1}) = \frac{z_1^6}{(1-z_1^{-1})(1-z_1^{-2}z_2)}$$
$$f(K_{V_2}) = \frac{z_1^2z_2 + z_1^2z_2^2 + z_1^3z_2}{(1-z_1^{-1}z_2^{-1})(1-z_1^2z_2^{-1})}$$

Summing these fractions and reducing yields indeed a sum over the monomials representing the integer points inside the triangle:

$$f(K) = f(K_{V_0}) + f(K_{V_1}) + f(K_{V_2})$$
$$f(K) = \frac{1}{(1-z_1)(1-z_1z_2)} + \frac{z_1^6}{(1-z_1^{-1})(1-z_1^{-2}z_2)} + \frac{z_1^2z_2 + z_1^2z_2^2 + z_1^3z_2}{(1-z_1^{-1}z_2^{-1})(1-z_1^2z_2^{-1})}$$
$$f(K) = 1 + z_1 + z_1^2 + z_1^3 + z_1^4 + z_1^5 + z_1^6 + z_1z_2 + z_1^2z_2 + z_1^3z_2 + z_1^4z_2 + z_1^2z_2^2$$

Thus the triangle given by V_0, V_1 and V_2 contains the following lattice points according to the function $f(K)$:

$$\{(0,0),(1,0),(2,0),(3,0),(4,0),(5,0),(6,0),(1,1),(2,1),(3,1),(4,1),(2,2)\},$$

which can also be verified with figure 2.1.

An evaluation of the above function $f(K)$ at the point $(z_1, z_2, \ldots, z_n) = (1, 1, \ldots, 1)$ yields the number of lattice points, since each monomial evaluates to 1 and summing over all points inside the given lattice yields the desired cardinality. Using this approach for the shorter notation with the sum of fractions is impossible, because then the denominator of each summand evaluates always to zero meaning that the function has a pole and is therefore not defined at this point. Nevertheless, with the help of methods from numerical complex analysis, the limit of the function around that point can be determined.

Several modifications and improvements were proposed afterwards. De Loera et al. [12] conducted an implementation called `LattE` along with various experiments and reported on the results. `LattE` is capable of counting lattice points in a space of up to approximately 30 dimensions. The runtime of the algorithm is significantly influenced by the number of dimensions and might in general already take quite long for such large spaces.

Chapter 3

Crowdsourcing Optimization Model

In this chapter the focus lies on introducing a model for an optimizer which considers queries with multiple relevant answers involved in a join operation. Access paths are used as an abstraction to encode different information sources along with their reliability and access costs. Trade-off solutions are required in this context, because the quality should be maximized while minimizing costs and latency.

First, the model is introduced and explained along with all the needed abbreviations. Then the underlying assumptions are listed and illustrated in details. The decision-making functions are described as techniques for aggregation to ensure certain levels of agreement between workers and therefore quality. The third section states the definition of the utility function for a given plan and how it is integrated into the optimizer. Finally, we look at the prediction model and possibilities for optimizations.

3.1 Optimization Model and Underlying Assumptions

This section introduces the problem along with terms and abbreviations used throughout the whole thesis. Then a formal description of the model is given and finally the assumptions underlying the introduced model are listed and explained.

3.1.1 Introduction

An optimizer in a traditional relational database management system aims at applying heuristics to find an access plan which minimizes the execution time or latency for a given query. In contrast to this one-dimensional optimization objective, crowdsourcing databases deal with a more complex problem, namely to maximize the quality while minimizing monetary costs and latency at the same time. This target definition asks for trade-off solutions, e.g. to maximize the quality while keeping costs and latency within certain given boundaries.

Latency requirements are difficult to accomplish when a crowdsourcing platform such as Amazon's Mechanical Turk is involved. The completion time for a batch of tasks is highly influenced by the number of available workers and by the attraction of individual questions to the community. The availability of workers depends for example on their time zones and their working hours. The attraction is influenced by the problem domain and the amount of money paid as a reward. In the following, the focus lies on the trade-off between cost and quality.

Dolati [5] suggests in his thesis an abstraction model for the optimizer of crowdsourced databases which is based on the abstraction of access paths known from traditional relational databases. His work focuses on crowdsourcing single values of single attributes, i.e. for each question exists exactly one correct answer and a set of wrong answers of a known or estimated size. The goal of this work is to apply the same concepts to a more general setting, i.e. to problems with a set of correct answers. Furthermore, we aim at crowdsourcing joining conditions between several such data sets.

As described in the previous chapter on related work, two different types of joins can be considered. One approach looks at joining existing data tuples, but the evaluation of the join predicate might need to involve the power of the crowd. The second method focuses on acquiring new data while performing the join implicitly in the crowd. In this work, we will elaborate more on the second type.

The notion of an access path in the crowdsourcing context denotes an (independent) source of information. Each such source has its own reliability, access costs and domain of adoptable values. An access plan $\mathcal{P}$ defines how many questions will be asked on each access path. Workers then have to be encouraged adequately by the system to use the assigned access path i for acquiring the desired information. A worker answers with a set of m elements denoted as a vote v. All votes acquired according to a given plan $\mathcal{P}$ form a vote set $\mathcal{V}_{\mathcal{P}}$.

The assumption is that a query and a set of applicable access paths with their properties is given by the schema designer or application developer, since the crowdsourced database is not able to infer this information automatically. Domain knowledge is in most cases essential to find an appropriate set of information sources and to estimate their parameters.

Example 3.1 illustrates the type of problems that we are addressing.

Example 3.1 (Vegetarian Meals in San Francisco) *Which vegetarian meals are served at which restaurants in San Francisco?*

The goal is to crowdsource a list of restaurants with the following attributes: name and website. For the ones considered to be relevant and therefore being picked by the aggregation strategy, the menu available on their website or in reviews on Foursquare should be crowdsourced in a second step. The following access paths can be used to acquire the data:

1. *Use the crowdsourced database on* `http://www.yelp.com`
2. *Use Foursquare for your search:* `https://foursquare.com`
3. *Search on Happy Cow's site:* `http://www.happycow.net`

3.1.2 Model Description and Formal Definitions

Our model conforms to the one suggested and introduced by Dolati [5] in many characteristics. However, to deal with the slightly different requirements for gathering sets of values instead of single attributes, some important changes are made. In the following, the properties of the model are described and the formal definition in the new environment is stated and explained in details.

For every query, a set of access paths is considered and specified by the schema designer which can be used to obtain the best possible answer for a given budget. Each path is associated with a bunch of information related to the usage and identification. Each access path has a name, a description, costs in terms of money that is paid to workers for giving an answer on that path, an error rate representing the path's reliability, the domain size and the number of correct elements in the domain.

The set of available access paths is denoted by $\mathcal{A}$. The number of paths in $\mathcal{A}$ is denoted by n, i.e. $|\mathcal{A}| = n$. Similar to [5, p. 14/15], an access path $i \in \mathcal{A}$ is represented by the following attributes and properties:

- **Access index** a_i: The list of attributes that need to be given in order to use this access path. It can be considered as the input format to access path i.
- **Properties**:
  - **Name** n_i: A unique name for the proper identification of the access path.
  - **Description** d_i: An instruction describing the usage of that access path to the workers.

  - **Cost** c_i: Monetary cost paid to a worker who is exploring that access path.
  - **Error rate** e_i: The probability of giving a wrong answer when using that access path.
  - **Domain size** s_i: Number of feasible answer elements in the domain of the access path: $0 < s_i$.
  - **Correct count** s_{ic}: Number of correct or relevant items in the domain of the access path, thus $0 \leq s_{ic} \leq s_i$ holds always. The number of wrong or irrelevant items in the domain can be calculated as follows: $s_{iw} = s_i - s_{ic}$.

- **Path result** r_i: The list of attributes that will be obtained after having asked workers to use this access path. Similar to the *access index* a_i denoted as the input format, the *path result* r_i can be considered as being the output format of access path i.

Some parameters are not associated with an access path, but nevertheless they might influence the outcome of our utility calculations considerably. For example the budget is related to plans and reflects the money available for a certain query execution. The available budget restricts the number of questions that can be asked for a given query and a given set of access paths with associated costs.

Each worker is asked to return a vote containing m distinct answer elements from the domain of the associated access path. The final result is supposed to contain approximately k elements that are picked after aggregating the workers' answers. The domain overlap size o specifies the number of elements that are in the intersection of all domains, i.e. $o = |\mathcal{D}_1 \cap \ldots \cap \mathcal{D}_n|$. The number of contained correct elements is denoted with o_c.

The following list contains a summary of all the parameters and properties of the model that are independent of a specific access path i.

- **Budget** b: The budget represents the money available for a certain query. It restricts the number of questions being asked.
- **Number of access paths** n: This parameter denotes the number of elements in $\mathcal{A}$, i.e. $|\mathcal{A}| = n$.
- **Number of domain elements** m **in a vote**: Each worker is supposed to answer with m distinct elements from the domain of the associated access path.
- **Result size** k: After aggregating the votes from different workers (see section 3.2), the top-k elements are returned as a result to the query.

Due to ties and different resolution strategies (see section 3.3), the effectively returned number of elements might be smaller or larger.

- **Domain overlap size o**: This property specifies the number of elements that are shared accross all access paths' domains, i.e.

$$o = |\mathcal{D}_1 \cap \ldots \cap \mathcal{D}_n|.$$

- **Number of correct elements o_c in domain overlap**: Each element in the previously described distinct set $\mathcal{D}_1 \cap \ldots \cap \mathcal{D}_n$ of size o is either correct or wrong. $o_c \leq o$ denotes the size of the subset containing only the correct elements. The size of the subset containing only the wrong values can be derived: $o_w = o - o_c$.

3.1.3 Assumptions

In order to calculate utilities along with the withdrawal probabilities of various vote sets, we make several simplifying assumptions. The first two are very similar to the work of Dolati [5]. The third one had to be adapted to the new scenario and the last one was added explicitly as an assumption, although the same holds for the previous model.

Workers give answers with varying reliability. Since we introduced the concept of access paths, each worker is supposed to use the access path assigned to the task. We assume that workers' answers are less correlated if different paths for finding the answers are used, while consulting the same source of information might likely yield the same potentially wrong answer. As a simplifying assumption, workers' answers are considered to be independent if they use different access paths. On the same access path i, workers are considered to share the error rate e_i. Thus, we make the following assumption:

Assumption 1 *Answers are correlated within an access path, but independent accross access paths.*

Phenomena observed in the answer distribution and ordering of crowdsourcing marketplaces include the so-called *list walking* as described in [21]. It was observed that workers often find lists with answers to a certain query on the internet and submit the answers in a related order. In case such a list contains a wrong or irrelevant item, probably most workers consulting that list are still going to submit it. Alternatively, as described by Dolati in [5], the result is expected to contain so-called *clusters of answers*. Correctly and wrongly given answers and therefore the reliability are assumed to be associated with the access path instead of individual workers. This simplification is summarized in the following assumption.

Assumption 2 *An error rate is associated with an access path and not with an individual crowd worker.*

Each worker is considered to return a vote v containing m distinct answer elements from the domain $\mathcal{D}_i$ (associated with access path i) in case the involved question statement asked to use access path i (see subsection 3.1.2). We consider the process of drawing each of the m distinct elements from the domain $\mathcal{D}_i$ as a two-stage process.

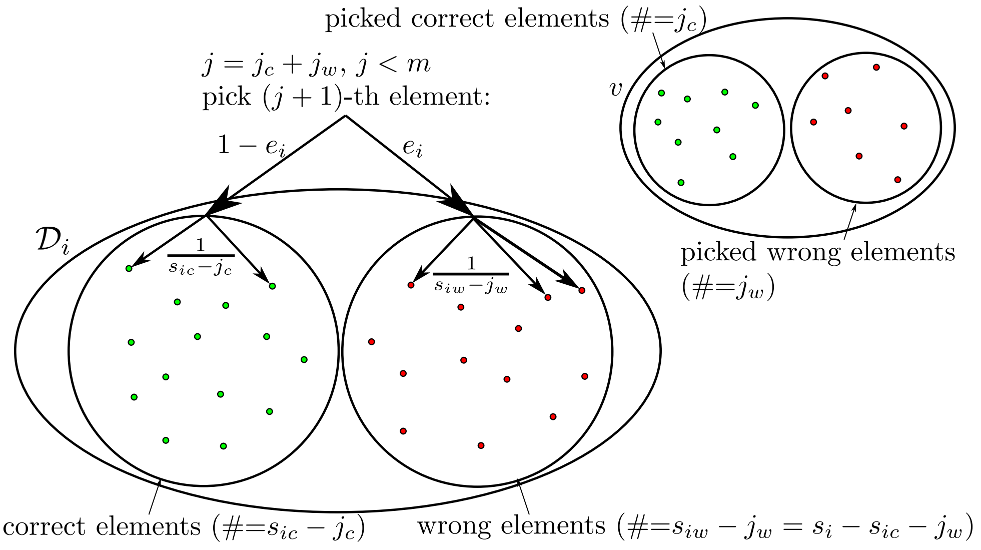

picked correct elements (#=j_c)
$j = j_c + j_w$, $j < m$
pick $(j+1)$-th element:
$1 - e_i$
e_i
v
$\mathcal{D}_i$
$\frac{1}{s_{ic} - j_c}$
$\frac{1}{s_{iw} - j_w}$
picked wrong elements (#=j_w)
correct elements (#=$s_{ic} - j_c$)
wrong elements (#=$s_{iw} - j_w = s_i - s_{ic} - j_w$)

Figure 3.1: Picking the $(j+1)$-th element from $\mathcal{D}_i$ for vote v

Two different cases for the first stage can be distinguished when calculating the withdrawal probability of a vote v:

- Both correct and wrong elements are remaining in the domain and deciding for one of the two subsets happens with probability $1 - e_i$ and e_i respectively. As depicted in figure 3.1, picking the $(j+1)$-th element after having picked $j < m$ elements for v gives for each remaining correct element a probability of $(1 - e_i) \cdot \frac{1}{s_{ic} - j_c}$ and for each remaining wrong element a probability of $e_i \cdot \frac{1}{s_{iw} - j_w}$.
- One of the two distinct subsets (correct / wrong elements) is exhausted after having picked $j < m$ elements for v and only the other one contains elements that can be returned by the worker. As depicted in figure 3.2, an element of the non-empty subset is returned as the $(j+1)$-th element with probability $1 \cdot \frac{1}{s_{ic} - j_c}$ or $1 \cdot \frac{1}{s_{iw} - j_w}$ respectively.

Picking one specific answer element from these distinct subsets without replacement is the second step. As mentioned above for the two cases, each element is assumed to be returned equally likely from the respective sub-

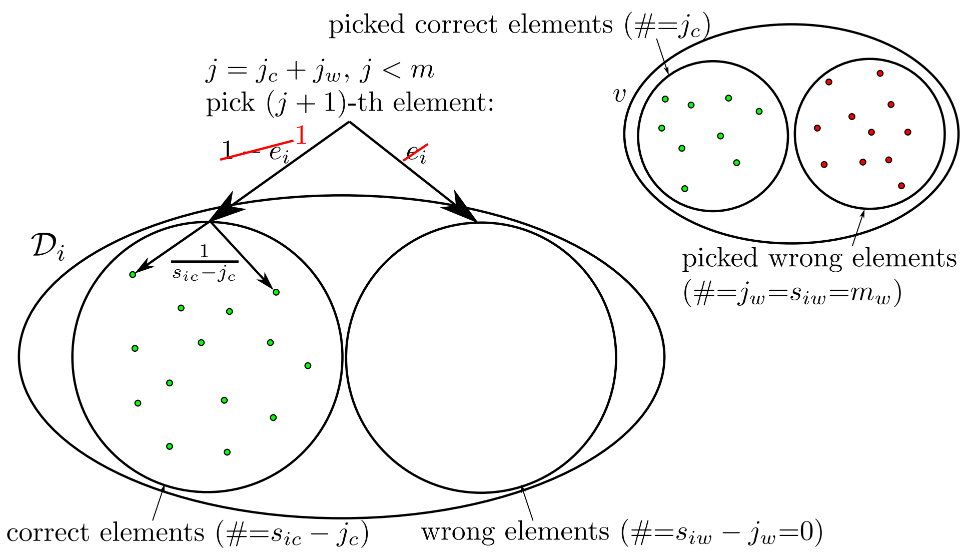


Figure 3.2: Picking the $(j+1)$-th element from $\mathcal{D}_i$ for vote v with only correct elements left

sets. This means that in each such step of withdrawing an element from the domain of the associated access path, a distinct correct element c_u is equally likely to be picked as another distinct correct element $c_v \neq c_u$. The same holds for the wrong elements respectively.

As can be seen in both of the above probabilities for picking the $(j+1)$-th element, the value depends on the number of previously picked correct and wrong elements j_c and j_w. The probabilities are therefore not constant while drawing the elements of vote v. The reason for non-constant probabilities is that the drawing is modelled to be executed without replacement which asks for adjustments in the respective probabilities from one drawing to the next.

In practice, distributions of elements might be highly skewed, dependent on the domain of the problem and the available access paths $\mathcal{A}$. Since a general model is derived here and the underlying distributions are usually not known in advance, we suggest to model the process of drawing elements from the domain without replacement as explained above.

The withdrawal probability for a given fixed vote v with $v_c < s_{ic}$ and $v_w < s_{iw}$ is calculated as follows:

$$P[v] = m!(1-e_i)^{v_c} e_i^{v_w} \cdot \frac{1}{s_{ic} \cdot \ldots \cdot (s_{ic} - v_c + 1)} \cdot \frac{1}{s_{iw} \cdot \ldots \cdot (s_{iw} - v_w + 1)} \tag{3.1}$$

$$= m!(1-e_i)^{v_c} e_i^{v_w} \cdot \frac{(s_{ic} - v_c)!}{s_{ic}!} \cdot \frac{(s_{iw} - v_w)!}{s_{iw}!} \tag{3.2}$$

where v_c denotes the number of correct elements in v, $v_w = m - v_c$ denotes the number of wrong elements in v, s_i is the domain size of access path i

(i.e. $s_i = |\mathcal{D}_i|$) and s_{ic} and $s_{iw} = s_i - s_{ic}$ are the number of correct and wrong elements in the domain $\mathcal{D}_i$ of access path i respectively.

Each factor in formula 3.2 can be explained as follows:

- $m!$: The factor $m!$ is used as the number of permutations for the m distinct elements in the set, since the ordering of the elements in v is not important (see example 3.2).
- $(1-e_i)^{v_c} e_i^{v_w}$: It denotes the first stage of the drawing, which gives more weight to the correct elements and less weight to the wrong elements according to the reliability and error rate of the access path.
- $\frac{(s_{ic}-v_c)!}{s_{ic}!} \frac{(s_{iw}-v_w)!}{s_{iw}!}$: This factor represents the drawing without replacement in the two subsets of correct and wrong elements respectively.

Example 3.2 *The following votes are all equal since the ordering of individual elements is not important:*

$$v = \begin{pmatrix} c_1 \\ c_2 \\ w_1 \end{pmatrix}, v = \begin{pmatrix} c_1 \\ w_1 \\ c_2 \end{pmatrix}, v = \begin{pmatrix} c_2 \\ c_1 \\ w_1 \end{pmatrix}, v = \begin{pmatrix} c_2 \\ w_1 \\ c_1 \end{pmatrix}, v = \begin{pmatrix} w_1 \\ c_1 \\ c_2 \end{pmatrix}, v = \begin{pmatrix} w_1 \\ c_2 \\ c_1 \end{pmatrix}$$

Therefore the factor $m!$ is used in the formula for the withdrawal probability of a vote v.

The following assumption summarizes the properties of the probability distribution as it will be used in subsequent sections:

Assumption 3 *The probability of a vote v containing m distinct answer elements from the domain $\mathcal{D}_i$ of access path i with v_c correct and $v_w = m - v_c$ wrong elements is defined as:*

$$P[v] = \begin{cases} m!(1-e_i)^{v_c} e_i^{v_w} \cdot \frac{(s_{ic}-v_c)!}{s_{ic}!} \cdot \frac{(s_{iw}-v_w)!}{s_{iw}!} & \textit{if } v_c < s_{ic} \wedge v_w < s_{iw} \\ \sum_{j=0}^{v_w} \binom{j+v_c-1}{v_c-1} (1-e_i)^{v_c} e_i^{j} \cdot \frac{1}{s_{ic}!} \cdot \frac{(s_{iw}-v_w)!}{s_{iw}!} & \textit{if } v_c = s_{ic} \wedge v_w < s_{iw} \\ \sum_{j=0}^{v_c} \binom{j+v_w-1}{v_w-1} (1-e_i)^{j} e_i^{v_w} \cdot \frac{(s_{ic}-v_c)!}{s_{ic}!} \cdot \frac{1}{s_{iw}!} & \textit{if } v_c < s_{ic} \wedge v_w = s_{iw} \\ 1 & \textit{if } v_c = s_{ic} \wedge v_w = s_{iw} \end{cases} \tag{3.3}$$

where v_c denotes the number of correct elements in v, $v_w = m - v_c$ denotes the number of wrong elements in v, e_i is the error rate, s_i the domain size, s_{ic} the number of correct elements and $s_{iw} = s_i - s_{ic}$ the number of wrong elements in the domain $\mathcal{D}_i$ of access path i.

As a last assumption and simplification of the model, let us consider the topology of the access path domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$. The overlap of domains between different access paths could be arbitrary, meaning that each element

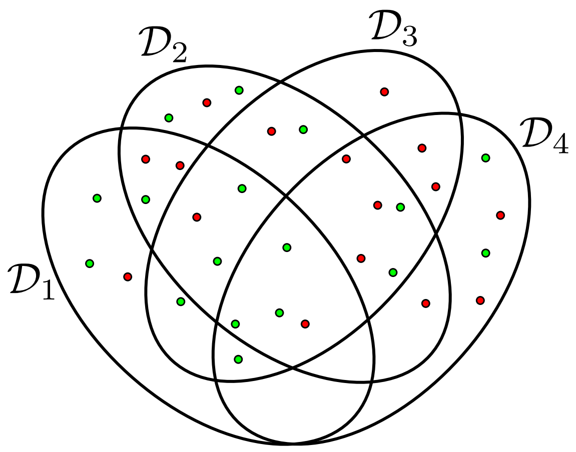


Figure 3.3: Arbitrary number of elements in each intersection of domains

of the power set of the domains can contain an arbitrary number of elements. Figure 3.3 illustrates that four domains have already fifteen intersections and thus an exponential number in the size of the number of domains and therefore in the number of access paths.

In practice, the structure of these domains and the sizes of intersections are unknown. Estimating an exponential number of sizes of intersections is clearly unfeasible. As a simplification, we therefore consider only domain topologies that satisfy the following assumption.

Assumption 4 *Domains are considered to be distinct except for one particular set of values which is shared across all domains of the different access paths. Its size and structure are specified by the domain overlap size $o = |\mathcal{D}_1 \cap \ldots \cap \mathcal{D}_n|$ and the number of contained correct elements $o_c \leq o$.*

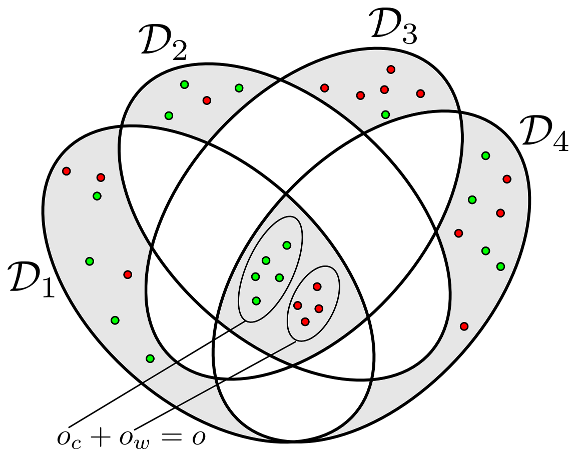


Figure 3.4: Distinct domains except the intersection $\mathcal{D}_1 \cap \ldots \cap \mathcal{D}_n$ can contain elements

This assumption makes it easier to instantiate our model, since the number of unknowns that need to be estimated is considerably lower. With this assumption, only the areas marked darker in figure 3.4 are considered to contain domain elements. Note that the number of wrong elements in the intersection can be determined as follows: $o_w = o - o_c$.

3.2 Decision-Making Functions

In this section we focus on the aggregation of various answers potentially obtained by exploring several different access paths. Workers on crowdsourcing marketplaces are in most cases not experts in the domain of the data and application. Quality answers are obtained by aggregating and combining several potentially differing answers in a meaningful way. This section is dedicated to three different decision-making techniques.

As described in subsection 2.2.2 of the previous chapter on related work, various crowdsourced databases such as *Qurk* and *Deco* allow user-defined functions and techniques for the aggregation of newly gathered data. The resulting flexibility of this approach is clearly desired, but on the other hand it becomes hard for the systems designer to know advantages and drawbacks of the different techniques. Thus, the goal of this section is to highlight and investigate the adapted versions of *Frequency Vote Strategy*, *Majority Vote Strategy* and *Likelihood Vote Strategy* as described by Dolati in [5] for the model of gathering data on single attributes.

Each worker is supposed to return a set of m distinct elements of the access path's domain $\mathcal{D}_i$ associated with the task, as described in a previous section. The input of a decision-making function consists of a parameter k and a set of answers from different workers each one acquired on a certain access path $i \in \mathcal{A}$ with $\mathcal{A}$ being the set of available paths for that query. As defined earlier, the plan $\mathcal{P}$ of a query specifies the number of questions being asked on each access path. The output consists of a set of elements from the union of all available access paths' domains. The input parameter k influences the size of this result set.

A more formal description of a decision-making function f as used in our context is depicted in figure 3.5 and can be described as follows:

$$f[k, \mathcal{A}, v_1, \ldots, v_p, path(v_1), \ldots, path(v_p)] = w$$

where

- $k \in \mathbb{Z}_+, k > 0$
- $\mathcal{A}$: the set of available access paths

- $path(v)$ denoting the index in $\mathcal{A}$ of the access path used for vote v; $path(v) \in \{1, \ldots, n\}$
- $\forall j \in \{1, \ldots, p\} : v_j \subseteq \mathcal{D}_{path(v_j)}, |v_j| = m$
- $w \subseteq \bigcup_{t=1}^{p} \mathcal{D}_{path(v_t)}, |w| \approx k$

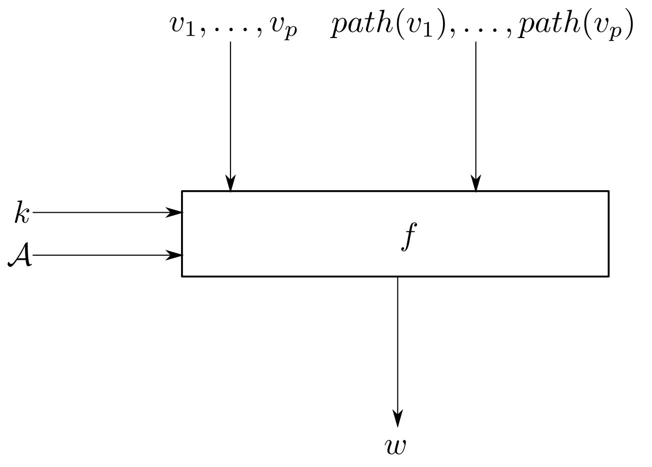


Figure 3.5: Formal description of a decision-making function f

3.2.1 Frequency Vote Strategy

The frequency vote strategy counts for each unique element in the union of the domains, i.e. $\bigcup_i \mathcal{D}_i$, the number of occurrences in all votes. The given threshold k is then used to pick the k elements with largest frequencies. Obviously, this method does not depend on the different access paths explored to acquire the votes. Therefore, the result of the decision-making function f is fully specified without $\mathcal{A}$ and without $path(v_1), \ldots, path(v_p)$ as input: $f(k, v_1, \ldots, v_p) = w$. Note that the reliability or error rate of each access path is not considered in this simple and straight-forward aggregation method.

Example 3.3 *Assume $p = 4$, $m = 4$, $k = 5$. The votes obtained from workers are:*

$$v_1 = \begin{pmatrix} c_1 \\ c_3 \\ c_4 \\ w_1 \end{pmatrix}, v_2 = \begin{pmatrix} c_1 \\ c_4 \\ w_2 \\ w_3 \end{pmatrix}, v_3 = \begin{pmatrix} c_1 \\ c_3 \\ c_5 \\ w_3 \end{pmatrix}, v_4 = \begin{pmatrix} c_3 \\ c_6 \\ w_2 \\ w_4 \end{pmatrix}$$

The frequencies of individual elements are therefore in decreasing order:

$$c_1 : 3 \quad c_3 : 3 \quad c_4 : 2 \quad w_2 : 2 \quad w_3 : 2 \quad c_5 : 1 \quad c_6 : 1 \quad w_1 : 1 \quad w_4 : 1$$

Picking the top-k with $k = 5$ yields as a result:

$$w = \begin{pmatrix} c_1 \\ c_3 \\ c_4 \\ w_2 \\ w_3 \end{pmatrix}$$

Note that in example 3.3 we are lucky that the top-k elements for $k = 5$ is unique. In case $k = 4$, it is not obvious which two of the three elements c_4, w_2, w_3 should be included together with c_1 and c_3 in the final result. The resolution of ties is treated in section 3.3 in more details.

3.2.2 Other Decision-Making Functions

Majority Vote Strategy The majority vote strategy is very similar to the frequency method, but the number of occurrences has to be at least $\lceil \frac{p}{2} \rceil$ for an element to be included in the final result w. Thus, if k is chosen to be sufficiently small, these two methods for resolution are identical. We do not consider this aggregation strategy any further in the experiments.

Likelihood Vote Strategy The likelihood vote method calculates the probability for each answer of being correct. Since our answers consist of sets, all possibilities of such sets would have to be considered in the computation. Additionally, we don't know which of the elements are correct and which are not within that answer set, so we have to enumerate all possible fractions of correct elements. Listing all these cases to calculate the likelihood score becomes very soon infeasible and is therefore not considered in the remaining parts of this thesis.

3.3 Resolution of Ties

As already mentioned in multiple other sections and example 3.3, the set of the k elements with highest frequencies might not be unique. In this subsection, the focus will be on various techniques to resolve ties.

Let us first describe the problem more formally: For the set of unique elements in all the domains, $\mathcal{U} = \mathcal{D}_1 \cup \ldots \cup \mathcal{D}_n$, the first step (denoted g) of the above-mentioned function f is to assign scores (or frequencies) to each element in $\mathcal{U}$:

$$g : \mathcal{U} \to \mathbb{R}_+.$$

As in the case of the frequency vote strategy, the function g is usually not injective, i.e.

$$\exists u_i, u_j \in \mathcal{U} : u_i \neq u_j \wedge g(u_i) = g(u_j).$$

The second step is to pick the k elements of $\mathcal{U}$ which have the highest values $g(.)$ for a given positive integer k.

Choose Random Tie Elements One possibility is to choose the number of needed elements from the subset of ties randomly with equal probabilities. The problem of this approach is that it is hard to reproduce utility calculations, because the number of correct elements in w might vary from one execution to another even for the exact same plan.

Exclude all Ties from Result Another possibility is to exclude all ties from the final result, thus returning a set w with size $|w| \leq k$. This method might return an empty answer, because more than k elements might have the same highest value among all frequencies. This is clearly undesired and should therefore be avoided.

Include all Ties in Result Similarly, instead of excluding the ties it is also possible to include them in the final result set w. The aforementioned problem of an empty result set does not occur any longer and that is the main reason why we used this method for the resolution of ties in the experiments. Note that it is essential to ask always at least two questions on a crowdsourcing platform in order to not only obtain frequencies of ones and therefore include all the elements contained in a vote.

3.4 Expectation Model

As highlighted by Dolati in [5, p. 23], the aggregation methods from section 3.2 are only applicable if the workers' votes are known in advance while the goal of the optimizer is to predict the number of questions asked on each available access path before the tasks are posted on a crowdsourcing platform. Therefore, the objective is to build an *expectation model*, which calculates the utility and hence the expected quality of different plans that are feasible for a given budget.

A plan $\mathcal{P}$ for a given maximum budget b is given as an array-like distribution of the votes over the available access paths $\mathcal{A}$. Each integral value $\mathcal{P}[i]$ denotes the number of questions being asked on access path i (see example 3.4). The goal is to assign a score to each plan which represents the expected utility in terms of the given access paths.

Example 3.4 *For $n = |\mathcal{A}| = 3$ and a predefined ordering of access paths in $\mathcal{A}$ a possible plan could be $\mathcal{P} = [3, 2, 0]$. This means that 3 workers are asked to use the first access path and 2 workers to use the second. The third path is not used with the above plan $\mathcal{P}$.*

Information retrieval uses two basic measurements for the quality of retrieved items, namely *precision* and *recall*. Precision denotes the fraction of retrieved items that are relevant to the user's information need. Recall refers to the fraction of relevant items in the whole collection that are retrieved as an answer. Information retrieval systems usually optimize for a high precision, since the number of retrieved items is in general too large to be processed by the user. Therefore, the following definition of the *utility* $U_{f,\mathcal{P}}$ uses precision as an objective to be maximized.

Definition 3.5 (Utility $U_{f,\mathcal{P}}$) *For a given plan $\mathcal{P}$ and a decision-making function f with a parameter k, the **utility** $U_{f,\mathcal{P}}$ is defined as a **random variable** denoting the fraction of relevant items in the result set w obtained from decision-making function f. The sample space $\mathbb{V}_{\mathcal{P}}$ contains all possible vote sets that can be obtained according to the given plan $\mathcal{P}$.*

$$U_{f,\mathcal{P}} : \mathbb{V}_{\mathcal{P}} \to [0, 1] \tag{3.4}$$

$$U_{f,\mathcal{P}}[\mathcal{V}_{\mathcal{P}}] = \frac{|\mathcal{U}_c \cap f[k, \mathcal{V}_{\mathcal{P}}]|}{|f[k, \mathcal{V}_{\mathcal{P}}]|} \tag{3.5}$$

$$U_{f,\mathcal{P}}[\mathcal{V}_{\mathcal{P}}] = \frac{|\mathcal{U}_c \cap w|}{|w|} \tag{3.6}$$

where $\mathcal{U} = \mathcal{D}_1 \cup \ldots \cup \mathcal{D}_n$ denotes the set of unique elements in all the domains and $\mathcal{U}_c$ is the set of all correct elements in all domains of the access paths in $\mathcal{A}$.

With each random variable a probability distribution is associated. In subsection 3.1.3, we described in formula 3.3 how the withdrawal probability for a vote v can be modeled. Since we assume statistical independence between different workers (see assumption 1), multiplying the withdrawal probabilities of all votes $v \in \mathcal{V}_{\mathcal{P}}$ obtained according to a plan $\mathcal{P}$ gives the withdrawal probability for a vote set $\mathcal{V}_{\mathcal{P}} \in \mathbb{V}_{\mathcal{P}}$.

$$P[\mathcal{V}_{\mathcal{P}}] = \prod_{v \in \mathcal{V}_{\mathcal{P}}} P[v] \tag{3.7}$$

Definition 3.6 ($score_{\mathcal{P}}$) *The **score** of a plan $\mathcal{P}$ is defined to be the **expected utility** $E[U_{f,\mathcal{P}}]$ which can be calculated as follows:*

$$score_{\mathcal{P}} = E[U_{f,\mathcal{P}}] = \sum_{\mathcal{V}_{\mathcal{P}} \in \mathbb{V}_{\mathcal{P}}} U_{f,\mathcal{P}}[\mathcal{V}_{\mathcal{P}}] P[\mathcal{V}_{\mathcal{P}}] \tag{3.8}$$

3.5 Prediction Models

It is assumed in our model that certain access paths in $\mathcal{A}$ are more reliable but also more expensive to explore than other less reliable and cheaper paths. A trade-off solution with a maximized quality for a given maximum budget b available for the query's execution is sought. Therefore, the goal is to find the plan $\mathcal{P}_{max}$ which has the highest score $\max_{\mathcal{P}} score_{\mathcal{P}}$ (defined as the highest expected utility $\max_{\mathcal{P}} E[U_{f,\mathcal{P}}]$) among all plans having costs smaller than or equal to the given budget b, i.e.

$$\sum_{i=1}^{n} \mathcal{P}[i] \cdot c_i \leq b \tag{3.9}$$

3.5.1 Enumeration of Plans and Vote Sets

The first suggested solution calculates the expected utilities accurately for each plan by first enumerating all the plans that are feasible with the given budget b and then enumerating all the possible vote sets for each plan according to the access paths and their properties. Although this implementation is pretty straight-forward, the problem is the exponential run time. Compared to the time needed for enumerating all vote sets $\mathbb{V}_{\mathcal{P}}$ for a given plan $\mathcal{P}$, the time needed to enumerate all the plans is negligible.

A domain size s_i for an access path i yields $\binom{s_i}{m}$ possible votes for one question being asked on that access path i, where m denotes the number of elements in each vote. The more questions are asked according to a given plan, the more such factors of binomial coefficients are needed to calculate the number of steps used in the vote set enumeration algorithm. The number of enumeration steps for a given plan $\mathcal{P}$ is:

$$\prod_{i=1}^{n} \binom{s_i}{m}^{\mathcal{P}[i]} \tag{3.10}$$

For each such step, the probabilities of all the votes in the enumerated vote set have to be multiplied to determine the vote set's probability as given in formula 3.7. Then the frequencies have to be gathered for all the elements in the set of unique elements $\mathcal{U} = \mathcal{D}_1 \cup \ldots \cup \mathcal{D}_n$, they have to be sorted decreasingly and afterwards the fraction of correct items in the set of top-k elements (after resolving ties, see section 3.3) is determined. Finally, the probability and the utility of the enumerated vote set are multiplied (denoted as *partial score*) and then added to the score calculated so far.

After having iterated over all vote sets, this sum of partial scores yields the overall score $score_{\mathcal{P}}$ as derived in formula 3.8. Such a large number of iterations together with the costly operations within each loop execution makes the calculations already infeasible for rather small problem instances. Many optimization strategies for still calculating the exact score $score_{\mathcal{P}}$ are considered in the next subsection.

3.5.2 Optimizations in Implementation

This subsection deals with three different types of optimizations related to the baseline implementation which was introduced in subsection 3.5.1.

Store votes and their probabilities The first improvement was to enumerate and store all $\binom{s_i}{m}$ feasible votes for each access path $i \in \{1, \ldots, n\}$ along with their probabilities given in formula 3.3 before calculating the partial scores for individual vote sets. This measure obviously avoids recalculating all the probabilities for each reoccuring vote and instead looks them just up in memory.

Storing even vote sets and their probabilities to reuse them for larger plans turned out to be difficult without filling up the whole memory. To follow this idea, one has to decide which enumerations should be kept and how they are combined in the subsequent enumerations of vote sets following more complex plans. Our considerations didn't go any further into that direction.

Use multithreading on multicore machine(s) A second optimization in the previously explained implementation was to use multiple threads for the calculation of several partial scores at the same time. They can be summed in each thread and at the end the results from all the threads are added to obtain the overall score $score_{\mathcal{P}}$. On a quad-core machine as it has been used for the experiments described in chapter 4, a multi-threaded solution returns considerably faster than the single-threaded one.

Replace the sorting with a more efficient selection The third possibility for improvement stems from the rather large time spent on sorting the domain elements according to their frequencies in every iteration. The more access paths used and the larger the associated domains, the more elements have to be sorted in every iteration of the above-mentioned vote set enumeration. Since we are only interested in a small and constant amount of elements, namely approximately k, it is more efficient to search for them instead of having the whole set in the desired order. To accomplish this, a special algorithm called *quickselect* was implemented.

Quickselect works in a similar way as quicksort, which was introduced in 1962 by Hoare [9]. First, a pivot is selected among all the elements and then the collection is divided into two subsets, one with elements smaller and another one with elements larger than the pivot element. Quicksort then continues recursively in both smaller sets with another pivot. For quickselect on the other hand, it is sufficient to continue in just one of the two subsets depending on k and the number of elements that are smaller than the pivot. As soon as the pivot has to be placed at position k, the algorithm terminates. To resolve ties, a linear scan gives all the elements with equal frequencies around the k-th element.

3.5.3 Mathematical Considerations for further Optimizations

This subsection deals with various considerations regarding a mathematical formulation of the problem and attempts to tackle it. The goal is to state all the conditions as a system of linear equations and inequalities and then find solutions for it.

Binary variables denote whether a certain element is picked or not for a specific vote. For each question, exactly m elements have to be picked from the domain $\mathcal{D}_i$ of the associated access path i. For question q being asked on access path i an equation with s_i newly introduced binary variables $S_\_/I_\_$ is stated as follows:

$$m = \overbrace{S_{1,q} + S_{2,q} + \ldots + S_{o,q}}^{S_\_\text{: shared elements in } \cap_t \mathcal{D}_t} + \overbrace{I_{1,q} + I_{2,q} + \ldots + I_{s_i-o,q}}^{I_\_\text{: elements only in } \mathcal{D}_i}$$

where the first o variables in the above equation denote the occurrence of elements shared accross all access paths' domains (named S for *shared*) and the last $s_i - o$ variables denote the occurrence of elements being specific to the domain of access path i (named I for access path i). It is not visible which variables denote correct or relevant items and which denote wrong or irrelevant items, as it is not needed here. Within the variables $S_{1,q}, S_{2,q}, \ldots, S_{o,q}$ exactly o_c denote correct elements and $o_w = o - o_c$ denote wrong elements. Within the variables $I_{1,q}, I_{2,q}, \ldots, I_{s_i-o,q}$, exactly $s_{ic} - o_c$ denote correct elements and $s_{iw} - o_w = (s_i - s_{ic}) - (o - o_c)$ denote wrong elements (see example 3.7).

Example 3.7 (m elements in each vote) *Let us assume that an access path i with $s_i = 8$ is given. Furthermore, $m = 5$ and $o = 5$ is known. Asking one question on that access path i yields the following equation:*

$$5 = \overbrace{S_{1,1} + S_{2,1} + S_{3,1} + S_{4,1} + S_{5,1}}^{S_\_\text{: shared elements in } \cap_t \mathcal{D}_t} + \overbrace{I_{1,1} + I_{2,1} + I_{3,1}}^{I_\_\text{: elements only in } \mathcal{D}_i}$$

where $S_\_$ and $I_\_$ are indicator variables, i.e. $S_\_, I_\_ \in \{0,1\}$.

The system of linear equations and inequalities has to be solved for each potential result set w. So the goal is to enumerate all possibilities how k elements (to consider ties even more) can be chosen out of the unique elements $\mathcal{U}$ of all the domains. For each such set, the smallest frequency (represented as a sum of all the indicator variables denoting the very same element) of the elements within w has to be larger than the largest frequency from the elements outside w (see example 3.8).

Example 3.8 (Frequency distribution yields desired result set) *For the illustration, a similar setup as in example 3.7 is used. Given is an access path* 1 *with* $s_1 = 8$ *and an access path* 2 *with* $s_2 = 6$, $o = 5$ *and a plan* $\mathcal{P} = [2, 1]$. *For a result set* w *containing the elements denoted with* $S_{1,\_}$, $S_{3,\_}$ *and* $I1_{1,\_}$, *the inequality is expressed as follows:*

$$\begin{aligned} &\min\{S_{1,1} + S_{1,2} + S_{1,3}, S_{3,1} + S_{3,2} + S_{3,3}, I1_{1,1} + I1_{1,2}\} \\ &> \max\{S_{2,1} + S_{2,2} + S_{2,3}, S_{4,1} + S_{4,2} + S_{4,3}, S_{5,1} + S_{5,2} + S_{5,3}, \\ &\qquad I1_{2,1} + I1_{2,2}, I1_{3,1} + I1_{3,2}, I2_{1,3}\} \end{aligned}$$

$I1\_$ *denote the elements which occur only in domain* $\mathcal{D}_1$ *and not in* $\mathcal{D}_2$ *and similarly,* $I2\_$ *denote the elements which occur only in domain* $\mathcal{D}_2$ *and not in* $\mathcal{D}_1$.

This can be rewritten as a system of linear inequalities, by stating that each of the arguments of the function min on the left hand side of the inequality has to be larger than each of the arguments of the max-function on the right hand side.

The first considerations around the above systems of linear equations and inequalities deal with finding a basis of solutions. Subsection 2.3.1 highlights the theory behind Hilbert bases. The problem is that such bases are very often doubly exponential in size and therefore in computation in the number of variables. Since the number of variables grows heavily with larger plans and domain sizes, these considerations can't be applied in our scenario.

The *Barvinok encoding* as explained in subsection 2.3.2 represents all latice points inside a convex polyhedron as a sum of monomials in a generating function. To calculate the number of lattice points, this formula can be used to determine the limit at a specific point, namely $z = (1, \ldots, 1)$. The goal of our work was to determine $z = (z_1, \ldots, z_d)$ such that the evaluation of the generating function as specified by Barvinok returns our desired expected utility.

As mentioned in subsection 2.3.2, this encoding can only be calculated in polynomial time for a fixed and rather small (i.e. $d \leq 30$ for current implementations, see [12, p.1287]) number of dimensions d. Since the number of variables grows quickly with increasing plan and domain size, Barvinok's encoding is infeasible to calculate for many given problem instances.

3.5.4 Optimization using Combinatorics

To avoid the fast growth in the number of variables for our system of equations and inequalities, another approach aims at encoding the frequencies in integral variables instead of using binary variables denoting the occurrence of a single element. Then the number of variables is bounded by the size of the set of unique elements in all the access paths' domains, i.e. $|\mathcal{U}|$. The question is whether all the conditions can still be expressed in a clear fashion and how the number of solutions to this system of equations and inequalities can be determined. Additionally, each solution of frequencies maps to different vote sets, which have to be considered to determine the probabilities.

Chapter 4

Experiments and Results

The goal of this chapter is to demonstrate our results based on simulated experiments as well as insights on real-world setups relying on Amazon Mechanical Turk. The first section highlights various experimental scenarios considering the interplay between the different parameters of the model. The second section presents how the model can be applied.

4.1 Simulated Experiments

This section presents the simulated experiments along with the according setups and results. As introduced in chapter 3, the set of available access paths with their respective properties such as cost, error rate and domain size and a set of global options like the maximum budget available, the number of items given by each worker (m) and the size of the final result (k) influence the outcome of the optimization model. In the following subsections, various scenarios are investigated in more detail. The impact of the most important parameters is visualized and explained.

Unless stated otherwise, the experimental results of the following subsections were obtained with a standard setup of the global parameters as follows:

- Budget $b = 5$
- Number of result elements $k = 4$
- Number of elements given by each worker $m = 3$
- Intersection size $o = 4$
- Number of correct elements in intersection $o_c = 2$

Two different configurations of access paths as listed in tables 4.1 and 4.2 were applied. For each experiment, we state the configuration in use along with the changes made to the various parameters.

Table 4.1: Configuration 1: Access paths with their properties

Index i	Cost c_i	Error rate e_i	Domain size s_i	Correct element count s_{ic}
1	1	0.10	8	4
2	1	0.10	8	4

Table 4.2: Configuration 2: Access paths with their properties

Index i	Cost c_i	Error rate e_i	Domain size s_i	Correct element count s_{ic}
1	3	0.10	12	6
2	2	0.20	10	6
3	1	0.25	8	4

In each experiment which uses configuration 1 as depicted in table 4.1, changes in a property associated with an access path are usually applied on the second access path's configuration. To highlight this, the subscript 2 is used for each variable as already introduced in the previous chapter, e.g. e_2 for the error rate of the second access path.

Configuration 1 uses a symmetric configuration, i.e. both access paths have the exact same properties and are therefore equally considered by the optimizer. For a variation of one property, we expect to observe which of the two access paths is favored according to the given configuration and how the expected utility develops over time. This setup facilitates the interpretation of the results.

4.1.1 Score based on varying Budget

In a first experiment, we investigate the difference of $score_{\mathcal{P}}$ for so-called pure and mixed plans. Pure plans have only one of the access paths available for acquiring data, while mixed plans are free to choose between any combination of information sources.

For the first experiment configuration 1 was applied without changing any properties of the first access path. The number of correct elements in the domain of the second access path was set to 3 (i.e. $s_{2c} = 3$) and to keep the number of elements in the domain the same, the number of wrong elements is increased by 1, i.e. $s_{2w} = 5$.

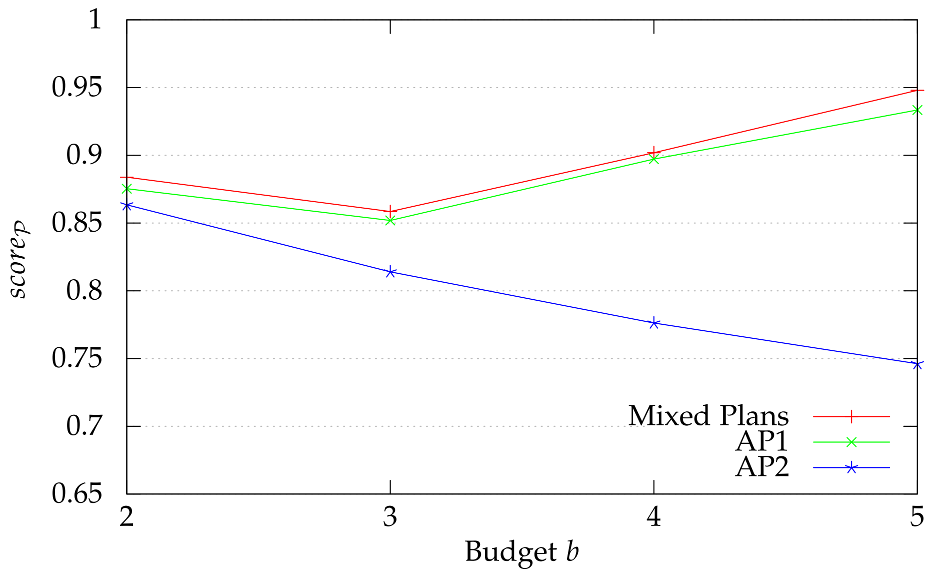


Figure 4.1: Mixed vs. pure plans for configuration 1

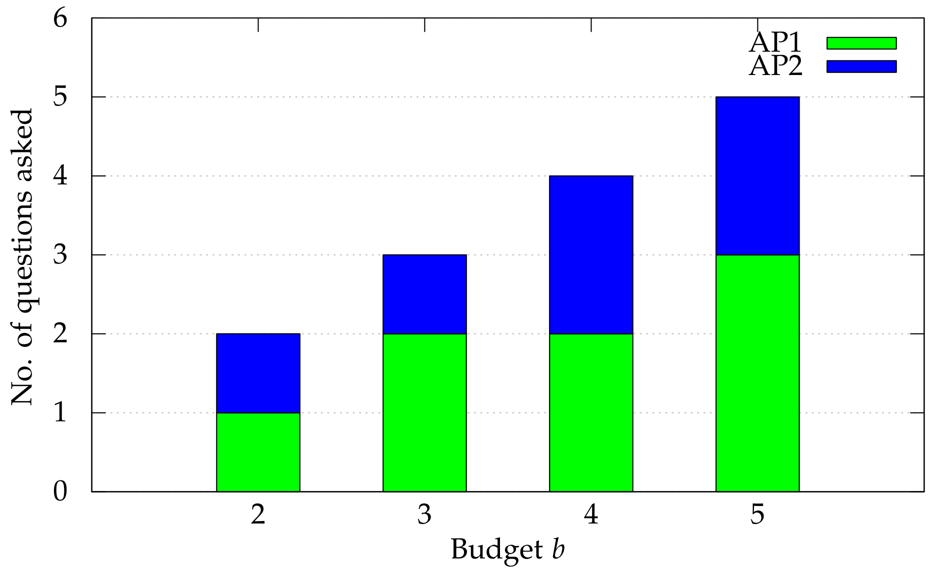


Figure 4.2: Mixed plans for configuration 1

Figure 4.1 shows the score for various budgets and the slightly changed configuration 1. Among the two pure plans, we expect to get a higher score using access path 1, since the number of correct elements in the domain is larger than on the second access path. This can easily be verified. The

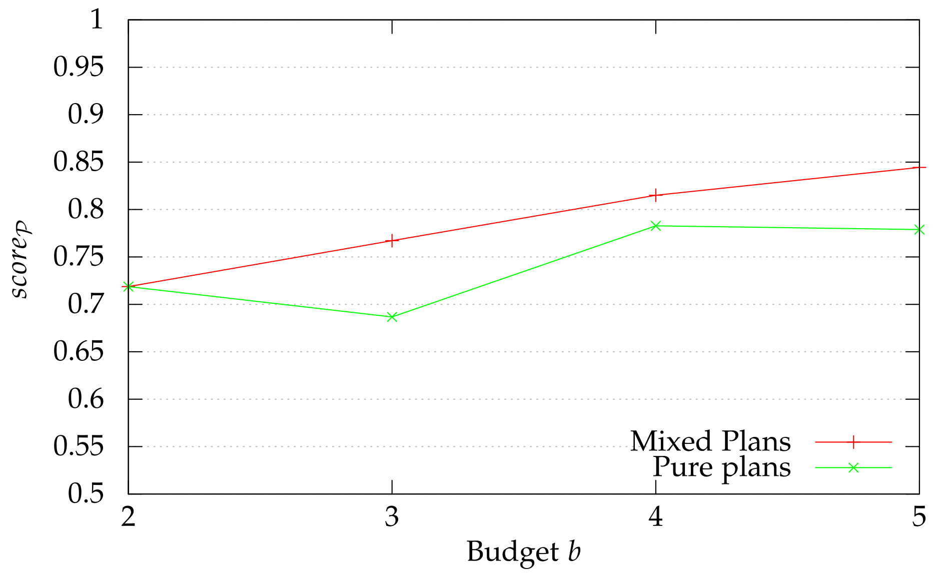


Figure 4.3: Mixed vs. pure plans for configuration 2

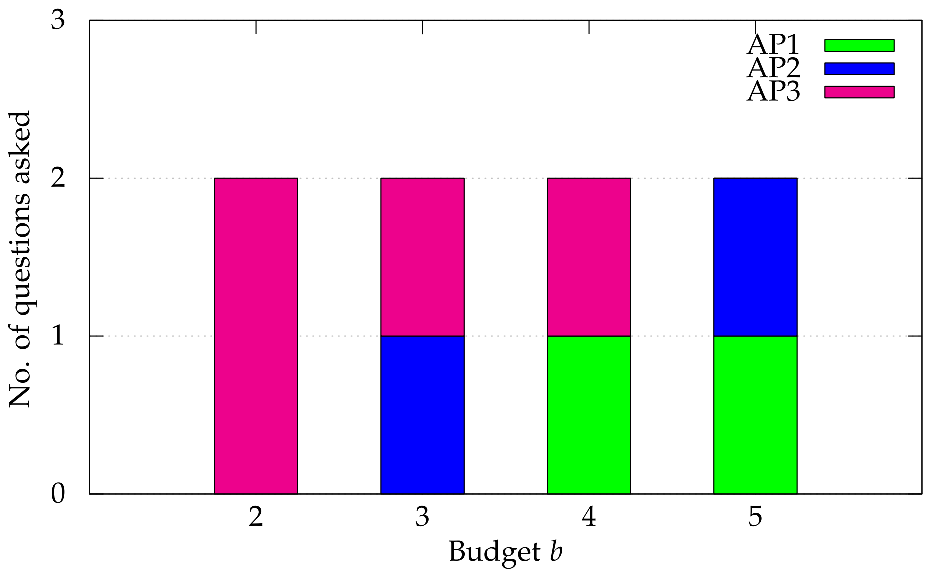


Figure 4.4: Mixed plans for configuration 2

best result is always obtained with mixed plans for various values of budget b. Therefore it pays off to use different sources of information to obtain a higher accuracy. Figure 4.2 shows the number of questions asked on each access path for the mixed plans yielding the highest scores in figure 4.1.

For the second configuration without having applied any changes, the trends of getting a higher accuracy for mixed plans is even stronger than with the previous setup (see figure 4.3). In this scenario, the optimizer can choose between 3 access paths instead of only 2. Therefore, the difference in our calculation of $score_{\mathcal{P}}$ between pure and mixed plans becomes even larger. Note that we always picked the pure plan with the highest score in figure 4.3 to compare with. Otherwise, the available budget b could for some access paths not be fully exploited, e.g. a budget of 5 allows to ask only 2 questions on the second access path with associated costs of 2 (i.e. $c_2 = 2$) yielding total costs of 4. Figure 4.4 shows again the number of questions being asked on each access path for the mixed plan of figure 4.3.

We have seen two rather balanced scenarios in this subsection where access paths imply higher costs the higher their provided accuracy is. The subsequent experiments show that for unbalanced scenarios with large differences in the error rates or very different domain structures, the superior access path is favored for the whole budget available. In theses cases, pure plans outperform mixed plans.

4.1.2 Score based on varying Number of Correct Elements

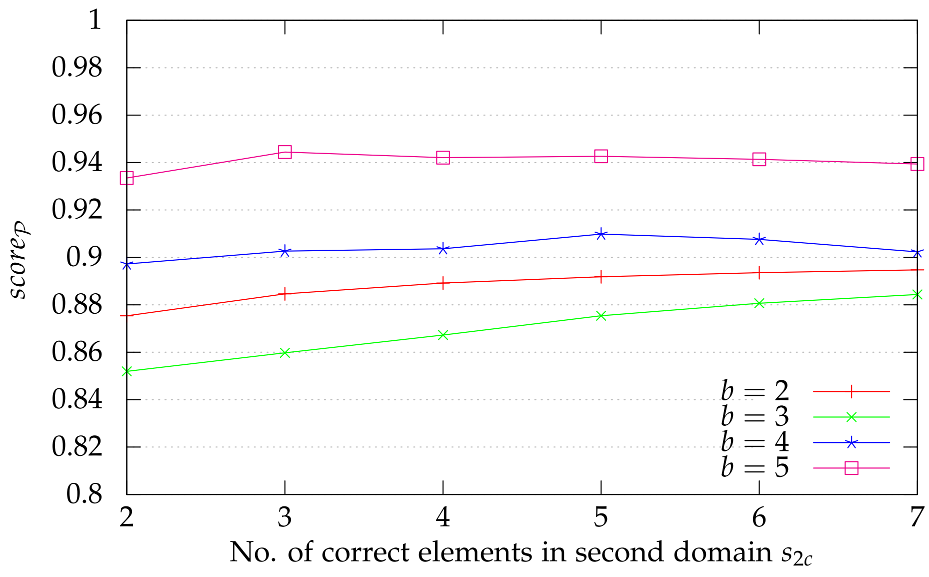


Figure 4.5: Increasing s_{2c}, keeping s_{2w} fixed

In these experiments, the goal was to find out how the number of correct elements in a domain influences the calculated scores. A legitimate concern

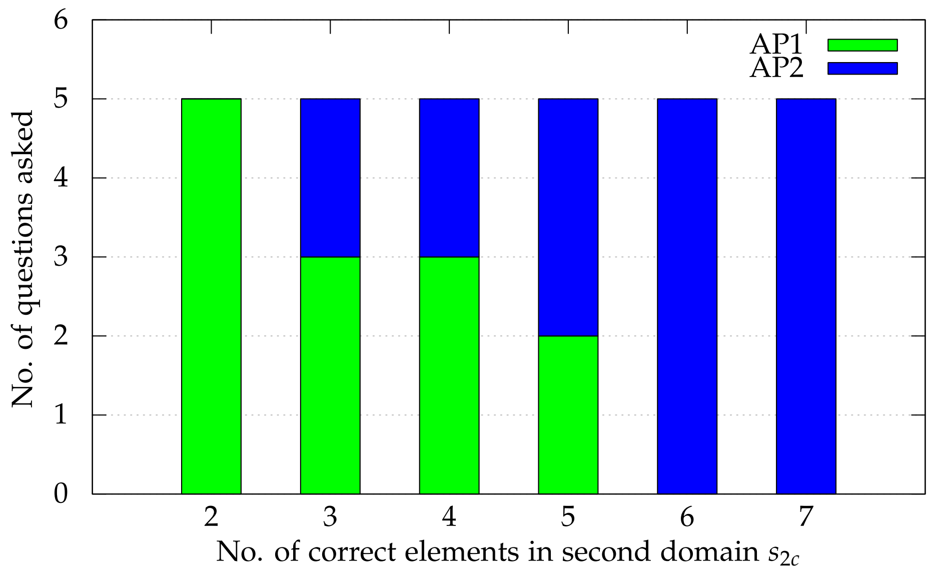


Figure 4.6: Plans for increasing s_{2c} (keeping s_{2w} fixed)

is whether we want to consider an increasing number of correct elements with a fixed number of wrong elements or with a fixed domain size. Keeping one of these two parameters fixed will automatically change the other one. Both of the two settings were investigated in our work and are presented in the following.

Figure 4.5 shows the scores obtained for an increasing number of correct elements in the domain of the second access path while keeping the number of wrong elements fixed with a value of $s_{2w} = 3$. This means that the domain size s_2 increases gradually with the increments of s_{2c}. In figure 4.6 the plans yielding the highest scores of figure 4.5 are shown. It is clearly visible that access path 2 is the preferred source of information over the other one if the difference in the number of correct elements is significant.

The second scenario is highlighted in figure 4.7. The number of correct elements in the second access path is increased and the domain size is fixed at $s_2 = 8$. Therefore with each added correct element, a wrong element is removed from the domain. Figure 4.8 shows the best plans for each of these domains. Having a very low number of correct elements and a large number of wrong elements on the second access path will clearly favor access path 1. If we add correct elements to the domain, it becomes more and more attractive to use this source of information. However, if only one or two wrong elements are left in the domain (i.e. $s_{2c} = 6$ or $s_{2c} = 7$), then they will have a relatively high probability for being selected. This is the case

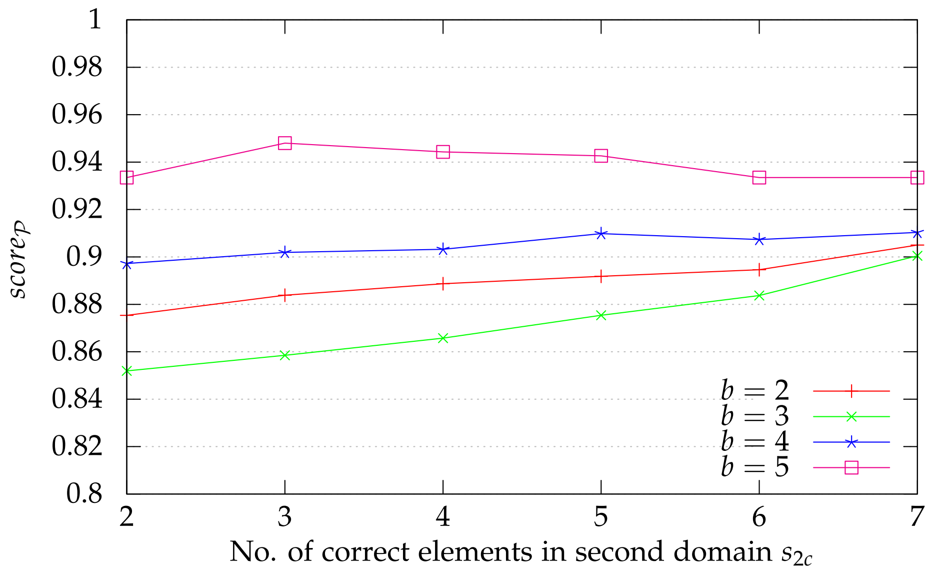


Figure 4.7: Increasing s_{2c}, keeping s_2 fixed

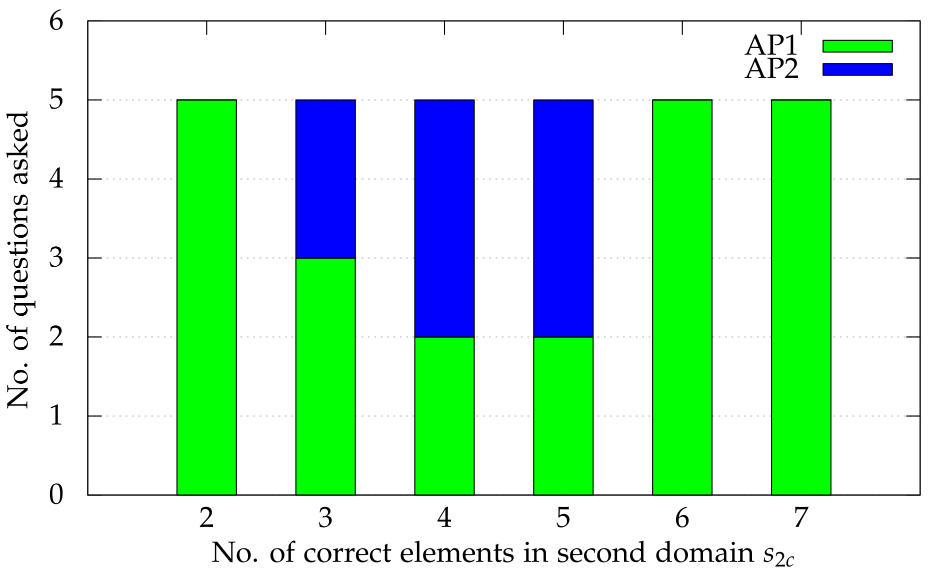


Figure 4.8: Plans for increasing s_{2c} (keeping s_2 fixed)

because we assume that one of the wrong elements is selected with a fixed probability given as the error rate e_2. That single wrong element is then more likely to be picked than picking one of the remaining correct elements.

4.1.3 Score based on varying Error Rates in one Access Path

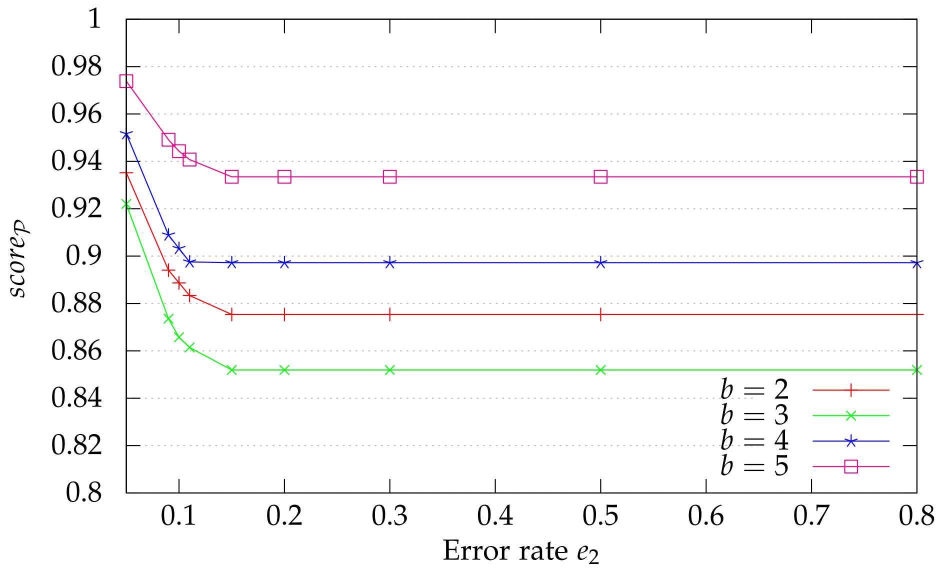


Figure 4.9: Increasing error rate e_2

This experiment shows the impact of various error rates e_2 in the second access path compared to the error rate of the first one for the configuration 1 as stated in table 4.1. The calculated scores plotted in figure 4.9 decrease for small values of the error rate e_2, but for values $e_2 \geq 0.2$ the score remains constant instead of showing the expected decay. This can be explained with the help of figure 4.10. Since the error rates and costs have a strong impact on which access path is chosen for each question, even for rather small differences in the error rates with $e_2 > e_1$ and same costs $c_1 = c_2$, only access path 1 is used and the error rate of the second access path has therefore no influence on the best plan's score.

4.1.4 Score based on varying Intersection Sizes

In this experiment, configuration 1 was used and the number of correct and wrong elements in the intersection has been increased in parallel, i.e. o_c and o_w were each incremented by 1, yielding an increment of 2 for $o = o_c + o_w$. The domain sizes of both access paths were kept constant with $s_1 = s_2 = 8$.

The result of increasing the size of the intersection of all the domains, i.e. the number of elements shared accross all domains $o = \cap_i \mathcal{D}_i$, yields for all the values of $b \in \{2, \ldots, 5\}$ a lower result in terms of the $score_{\mathcal{P}}$ as depicted in

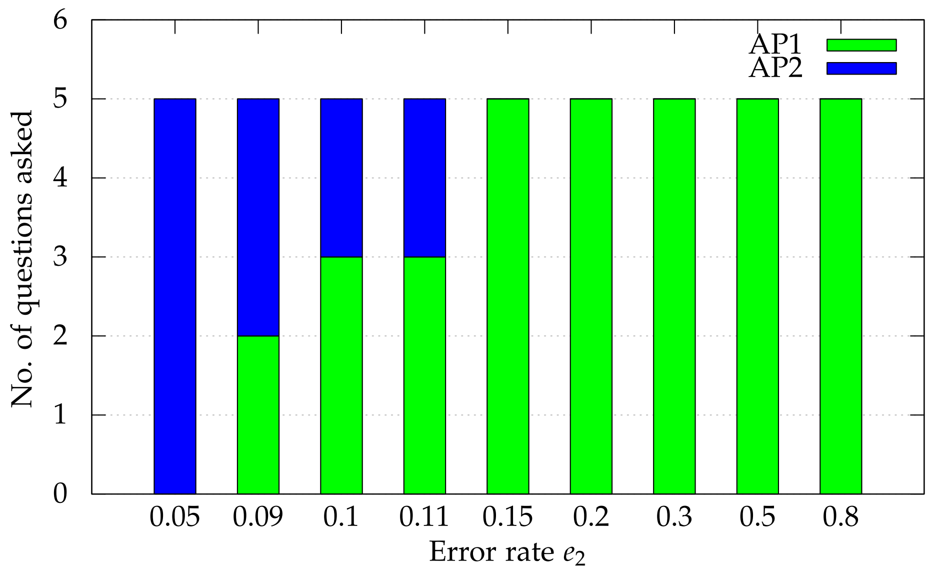


Figure 4.10: Plans for an increasing error rate e_2

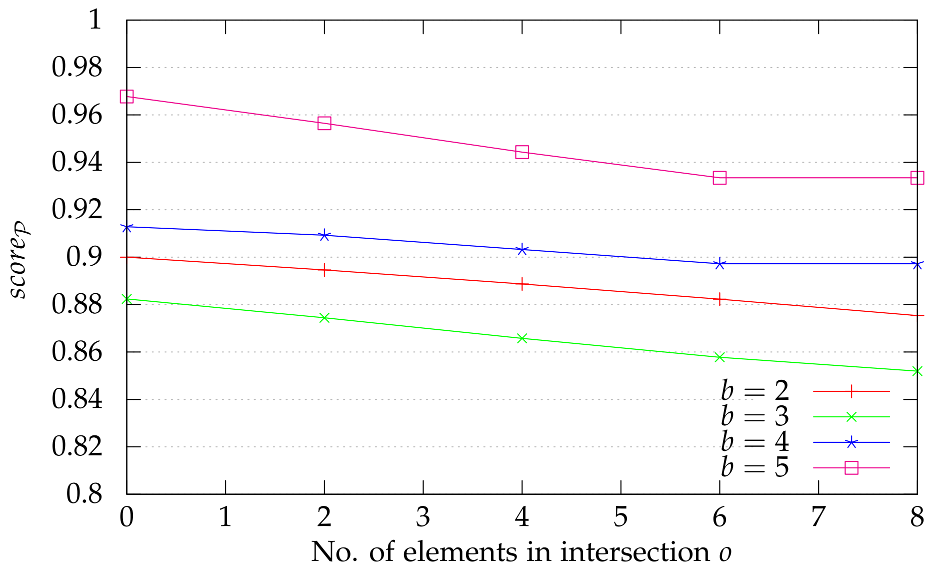


Figure 4.11: Increasing intersection size o

figure 4.11. A possible reason for this is that the distribution of frequencies changes, i.e. the elements of the intersection have higher frequencies and other elements lower ones, and because we include all ties in the final result, the overall expected utility decreases for a larger intersection.

So this demonstrates that there is less incentive to use similar access paths, because it becomes more unlikely to retrieve previously unseen elements.

4.1.5 Score based on varying Values for m

To see the impact of an increased number of unique elements m required in a worker's answer, configuration 2 was used. The general parameters stay the same as in the beginning of this section (see table 4.2), except for the value of m which is investigated.

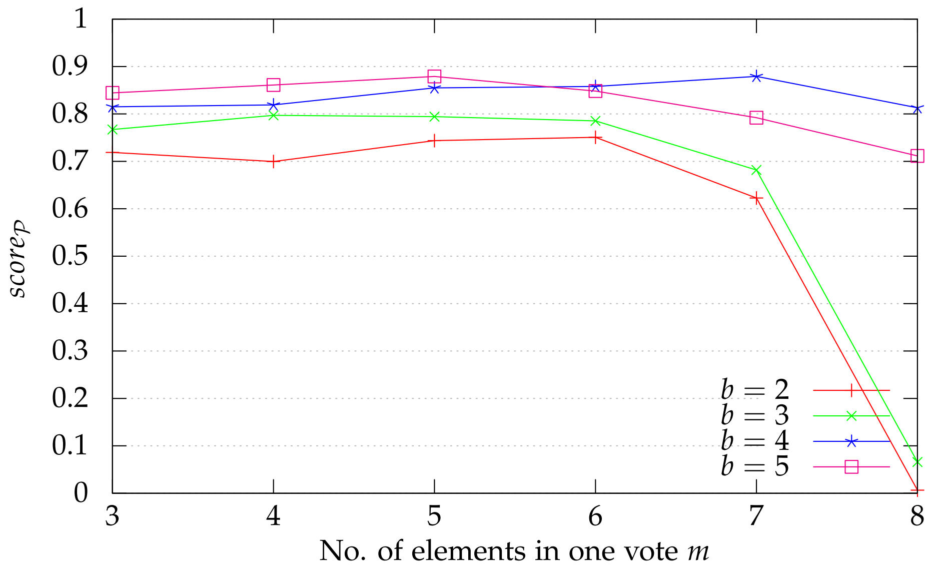


Figure 4.12: Increasing number of elements in a vote m

Figure 4.12 shows the impact on the expected utility $score_{\mathcal{P}}$ when increasing the number of elements in each worker's answer for several values of the budget b. If m is larger than the number of correct elements s_{ic} in the domain $\mathcal{D}_i$ of an access path i, then with each drawing at least some (i.e. at least $m - s_{ic}$) of the wrong elements have to be picked by a worker accessing that access path i. In case that the number of wrong elements s_{iw} in the domain is rather small, then it is more likely that the different workers will agree on a wrong element. The expected utility of the plan drops therefore significantly in that case.

4.1.6 Score based on varying Values for k

In the last two simulated experiments, the impact of varying the minimally required number of elements in the result set k was analyzed. The same setup as for varying m was used, i.e. configuration 2 as listed in table 4.2.

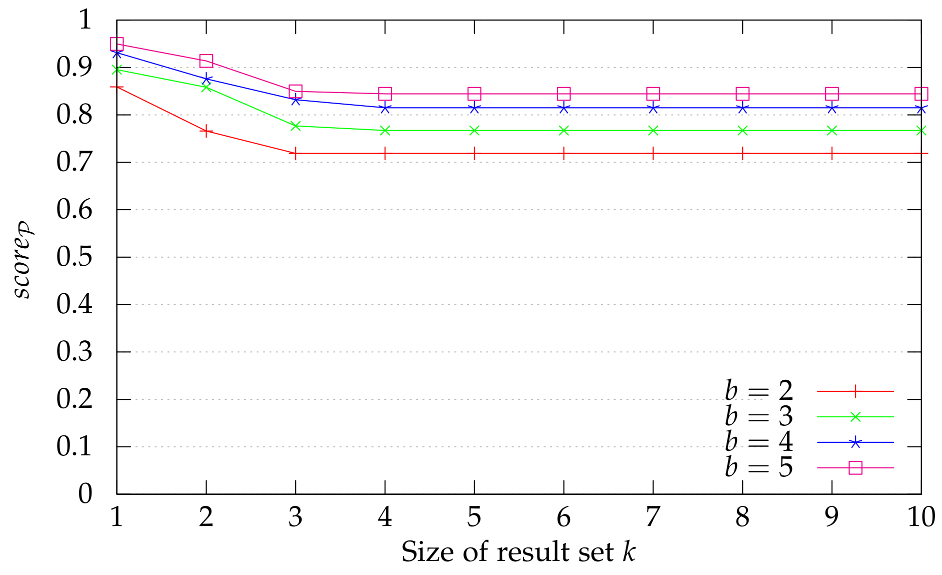


Figure 4.13: Increasing size of result set k ($m = 3$)

When looking at figure 4.13, we realise that the highest expected utility is reached for a small value of k, e.g. $k = 1$. In that case, only the most frequent element is returned as a result. With a high probability, this value will be a correct element, assuming that error rates are in general below 0.5. The more elements are added to the result set, the higher becomes the probability of adding a wrong element, since these occur less often and have therefore on average lower values in the frequency distribution.

Another issue attracting the attention is the constant utility for $k \geq 4$. It turns out that we used a rather small value for m, namely $m = 3$. The plan with the highest utility is already for a small value of k equal to $\mathcal{P} = [1, 1, 0]$ and since asking only one question on access path 1 and one question on access path 2 gives two answers containing each $m = 3$ elements from different domains, the expected utility is not influenced when increasing k. Note that we included all ties in the spectrum of frequencies among the top-k elements for each vote set enumeration and asking only one question on the first two access paths yields many ties in the resulting frequency distributions.

So we repeated the same experiment for a different value of m, namely

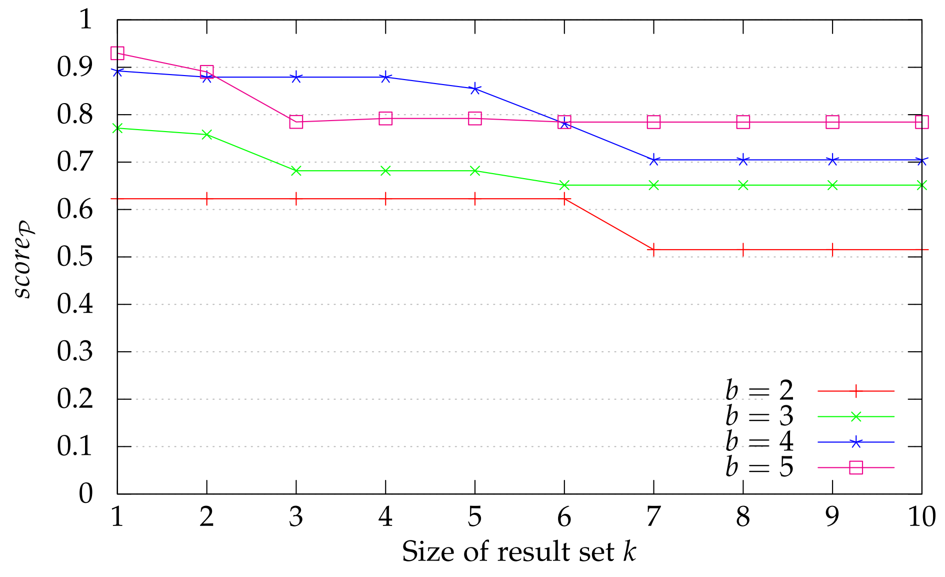


Figure 4.14: Increasing size of result set k ($m = 7$)

$m = 7$, as shown in figure 4.14. It turns out that the frequencies have more diverse values (since more elements can be aggregated) and therefore k is more relevant than for a smaller value of m.

4.2 Real-world Experiments using Amazon Mechanical Turk

To conclude this chapter on experimental results, a description of the tasks issued on Amazon Mechanical Turk along with the addressed challenges and results is presented in this section.

The goal was to crowdsource first a list of restaurants providing vegetarian meals, aggregate them and then crowdsource for the most frequently mentioned restaurants the meals they offer (see example 3.1). Obviously, the second batch of experiments (outer join relation) which acquires the meals needs the results of the first batch yielding the restaurants (inner join relation), so they cannot be performed in parallel. But within each batch, the answers for all the different access paths can be gathered alongside each other.

Example 3.1 describes the access paths available for this task, namely the following three crowdsourced databases:

1. Yelp: http://www.yelp.com

2. Foursquare: https://foursquare.com
3. HappyCow: http://www.happycow.net

HappyCow is domain-specific to vegetarian stores and restaurants and is therefore considered to yield quite accurate results. HappyCow does not offer the menu associated with restaurants, so we decided to use only the other two access paths for the second batch.

An interesting observation is that with the above list of access paths, the workers are guided to access information which was already crowdsourced before and collected on these sites. Therefore, the notion of an access path could also be viewed as a means to retrieve information acquired by a certain community of individuals and thus to provide access to a crowd on its own.

We asked the workers to provide 5 restaurants (name and URL) and 5 meals in each HIT respectively, i.e. $m = 5$ in both cases. To tune the parameters of our model such as the domain size and error rate, we decided to issue 15 assignments per HIT for the first batch which represents the plan $\mathcal{P}_1 = [15, 15, 15]$ and 10 assignments for the second batch, i.e. $\mathcal{P}_2 = [10, 10, 10]$. To investigate the behavior of the crowd and the distribution of answers, each of the previously described setups was repeated for the following five cities: New York, Chicago, San Diego, London and Amsterdam. The costs for the first batch can therefore be calculated as follows:

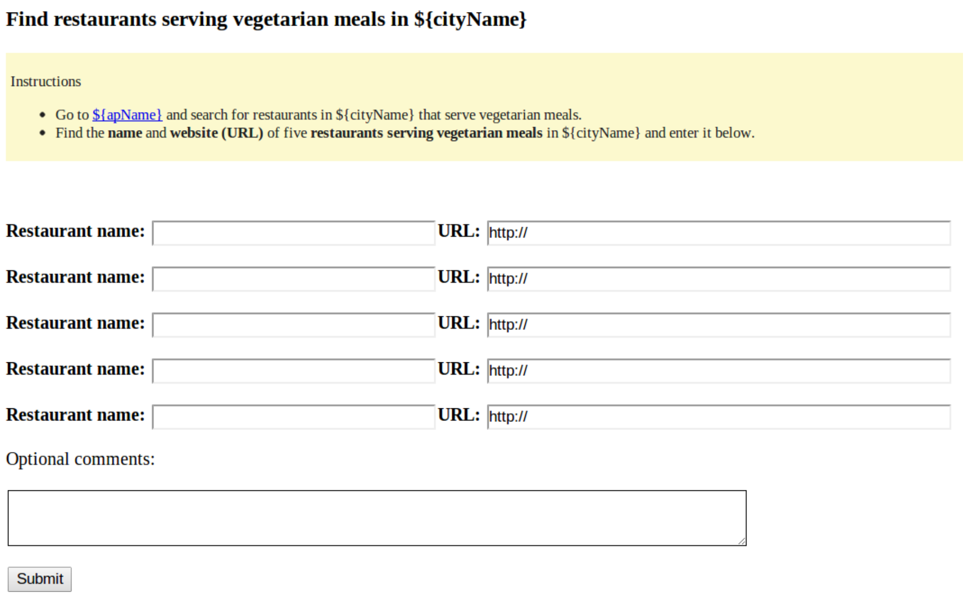
$$\begin{aligned} \text{costs} &= 1.1 \times 5 \text{ cities} \times 3 \text{ APs} \times 15 \text{ assignments} \times \frac{\text{price}}{\text{assignment}} \\ \text{costs} &= 247.5 \times \text{reward paid} \end{aligned}$$

where the factor 1.1 is used to add the 10% of fees that have to be paid to Amazon. A first challenge was to fix the price paid to workers for each assignment. We started with \$0.05, but no answers were provided within more than 12 hours. The same happened when offering \$0.10 for each assignment. Right after starting the whole batch another time with \$0.20, we realized that the workers were required to be qualified as so-called *masters*, which can ask for higher rewards in order to accept a task. This setting was provided by default and hidden in the configuration. Therefore it took a while to figure this out. The reward paid to workers without any qualification was finally \$0.10 for each assignment that asked for restaurants. The final costs for the first batch amounted to \$24.75.

The costs for the meals can be calculated in a similar way, but as stated earlier, only 2 access paths and 10 assignments were used for the second batch. Another factor has to be considered here: accessing the meals is done for each restaurant which was selected after aggregation (i.e. k of first batch). We used 5 restaurants in each city for the second batch. The reward per assignment was lowered to \$0.05.

$$\begin{aligned}\text{costs} &= 1.1 \times 5 \text{ cities} \times 2 \text{ APs} \times 10 \text{ assignments} \times 5 \text{ restaurants} \times \$0.05\\ \text{costs} &= \$27.50\end{aligned}$$

The general problem of cleaning the collected data (i.e. entity resolution) was addressed in many research projects as mentioned in chapter 2. For our example, the cleaning of the data was executed manually in order not to introduce any errors or uncertainties with automatic techniques.



Find restaurants serving vegetarian meals in ${cityName}

Instructions

- Go to ${apName} and search for restaurants in ${cityName} that serve vegetarian meals.
- Find the **name** and **website (URL)** of five **restaurants serving vegetarian meals** in ${cityName} and enter it below.

Restaurant name:		**URL:**	http://
Restaurant name:		**URL:**	http://
Restaurant name:		**URL:**	http://
Restaurant name:		**URL:**	http://
Restaurant name:		**URL:**	http://

Optional comments:

Figure 4.15: Form provided to workers to crowdsource restaurants

Figure 4.15 shows the html-form that was used in the first batch. The variables such as *£{cityName}* are automatically replaced with the values provided in a csv-file. The results of the top-5 restaurants for each city along with their frequencies are listed in table 4.3. Note that we got a tie for Amsterdam, where *TerraZen Centre* and *Superfood Centre at Health* were both provided 8 times. Thus the costs for gathering the meals are a bit higher than calculated previously.

Table 4.4 shows the top-5 results for each access path for Amsterdam. In comparison with the aggregated values in total as listed in table 4.3, it becomes evident that collecting only information in one access path might miss important elements for the overall result. Additionally, some elements have average values in all the access paths' frequency distributions, but on aggregate over all access paths their values are picked for the final result, thus the evidence for those becomes stronger.

Table 4.3: Top-5 restaurants crowdsourced on AMT according to example 3.1

f	New York	f	Chicago
22	Dirt Candy	26	Native Foods Cafe
20	Blossom Vegan Restaurant	19	Green Zebra
12	Red Bamboo	18	Mana Food Bar
9	TAIM	19	The Chicago Diner
9	Maoz Vegetarian	13	Quesadilla La Reyna del Sur
f	San Diego	f	London
28	Jyoti-Bihanga	20	Mildreds
28	Loving Hut	18	Tibits
25	Sipz Fusion Cafe	14	Food For Thought
16	Evolution Fast Food	13	The Gate
13	Casa de Luz - San Diego	12	Maoz
f	Amsterdam		
31	Maoz Vegetarian		
17	Golden Temple		
13	De Waaghals		
10	Betty's Vegetarisch Restaurant		
8	TerraZen Centre		
8	Superfood Centre at Health		

It turned out that the domain sizes of the three different access paths observed in this experiment were quite diverse. *Yelp* had in most cases a rather small number of elements compared to the other two. For example the observed domain sizes for the experiment asking for restaurants in San Diego were as follows: 31 for Foursquare, 39 for HappyCow and only 20 for Yelp.

Further investigations and assumptions are needed to estimate the error rate and domain sizes of individual access paths. A follow-up work might go into more details and verify the suggested model with real-world data.

Table 4.4: Restaurants crowdsourced for Amsterdam listed for each access path separately

f	Yelp	f	HappyCow
8	De Waaghals	13	Maoz Vegetarian
7	Golden Temple	8	TerraZen Centre
6	Maoz Falafel	6	Superfood Centre at Health
5	Marits huiskamer restaurant	4	Onkruid - Pop-up
3	Surya	4	Candy Freaks
3	Four Seasons Natural Foods	4	De Waaghals

f	Foursquare
12	Maoz Vegetarian
7	Golden Temple
6	Betty's Vegetarisch Restaurant
4	Soenda Kelapa
3	Superfood Centre

Chapter 5

Conclusion and Future Work

This final chapter of the thesis summarizes our conclusions in the first section and then provides an outlook for potential tracks to follow for future projects in this research area.

5.1 Conclusion

The motivation of the problem investigated in this thesis was to introduce a model which can be applied in an optimizer of a crowdsourced database that supports human-powered data gathering and joining. The notion of an access path as an independent source of information in the crowdsourcing context is the basic abstraction underlying our suggested model and denotes the degree of freedom for the optimizer. The goal is to find the best plan denoting how many times each source is accessed given certain parameters.

The challenge of a multidimensional optimization function which demands trade-off solutions in terms of monetary costs paid to workers, latency related to the time needed for the acquisition of information and accuracy of the final result is addressed. We suggest an abstraction model that consists of different parts such as an optimization model, an expectation model and a prediction model.

Our simulation experiments clearly show that for balanced scenarios where more reliable access paths are also more costly and the domains have a reasonable size, mixed plans are favored compared to pure plans. This verifies our expectation of reaching a higher accuracy when several information sources are combined to obtain the final answer. If one access path is clearly superior to the other ones, our model is assumed to allocate the whole budget on that single access path to obtain the highest quality. We verified this by increasing the error rate of one access path while all the other parameters remain the same.

An increasing number of correct elements makes the associated access path also more attractive to be integrated in the optimal plan, but only in case the number of wrong elements is not lowered at the same time (i.e. keeping the domain size fixed). This would change the probability distribution of our model such that the few wrong elements get a rather high probability of being included in the result. The parameter m defines the nature of the frequency distribution and k defines the threshold applied to this distribution. Since ties are always included in the final result, both parameters depend on each other.

The experiments on Amazon Mechanical Turk showed that accessing only a single access path might miss important values for the final result. Some elements are added to the result set, because their summed frequencies suggest to do so, even though each single access path would not include the element into the final answer. Thus, the join queries also need several access paths in their optimal plan to gather data that might otherwise be missing.

In summary, the presented model meets our expectations in terms of how the various parameters influence the structure of the optimal plan which is finally selected. The approach taken in this work offers many interesting directions for future work.

5.2 Future Work

Approximation One possibility for a follow-up project is to find an approximative solution for the computation of the expected utility which has about the same characteristics as our model, but is easier to calculate (not exponential runtime) and allows therefore to simulate the results for larger domain sizes and higher budgets. A promising approach could be to consider submodular set functions and based on their properties, derive a greedy algorithm.

Relaxing Assumptions and Refining Model Another idea is to relax the rather strong assumptions in our model based on insights gained from further experiments on Amazon Mechanical Turk or other crowdsourcing marketplaces. If the interdependencies between the various parameters are studied in more detail, a refinement of the model might predict the optimal plan more accurately.

Integration into Crowdsourcing Database The optimization model has been implemented as a standalone application and has therefore not been tested in the interaction with other important attributes and requirements of an optimizer for a crowdsourcing database. This offers another possibility for future work.

Bibliography

[1] Alexander I. Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. *Math. Oper. Res.*, 19(4):769–779, November 1994.

[2] Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C. Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samual White, and Tom Yeh. Vizwiz: nearly real-time answers to visual questions. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pages 333–342, New York, NY, USA, 2010. ACM.

[3] D. Chubarov and A. Voronkov. Basis of solutions for a system of linear inequalities in integers: computation and applications. In *Proceedings of the 30th international conference on Mathematical Foundations of Computer Science*, MFCS'05, pages 260–270, Berlin, Heidelberg, 2005. Springer-Verlag.

[4] Jesús Antonio De Loera. Generating functions algorithms in integer optimization. Slideshow, 2008. Available online at `https://www.math.ucdavis.edu/~deloera/TALKS/genfctslectures.pdf`; visited on August 14th 2013.

[5] Erfan Zamanian Dolati. *Query Optimization in CrowdDB*. Systems Group, Department of Computer Science, ETH Zurich, 2012.

[6] Amber Feng, Michael J. Franklin, Donald Kossmann, Tim Kraska, Samuel Madden, Sukriti Ramesh, Andrew Wang, and Reynold Xin. Crowddb: Query processing with the vldb crowd. *PVLDB*, 4(12):1387–1390, 2011.

[7] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: answering queries with crowdsourcing.

In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 61–72, New York, NY, USA, 2011. ACM.

[8] R. Hemmecke. On the Computation of Hilbert Bases and Extreme Rays of Cones. *ArXiv Mathematics e-prints*, March 2002.

[9] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

[10] Yannis E. Ioannidis. Query optimization. *ACM Comput. Surv.*, 28(1):121–123, 1996.

[11] Donald Kossmann and Konrad Stocker. Iterative dynamic programming: A new class of query optimization algorithms. *ACM Trans. on Database Systems*, 25:2000, 1998.

[12] Jesús A. De Loera, Raymond Hemmecke, Jeremiah Tauzer, and Ruriko Yoshida. Effective lattice point counting in rational convex polytopes. *J. Symb. Comput.*, 38(4):1273–1302, 2004.

[13] Adam Marcus, Eugene Wu, David Karger, Samuel Madden, and Robert Miller. Human-powered sorts and joins. *Proc. VLDB Endow.*, 5(1):13–24, September 2011.

[14] Adam Marcus, Eugene Wu, Samuel Madden, and Robert C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214. www.cidrdb.org, 2011.

[15] Priti Mishra and Margaret H. Eich. Join processing in relational databases. *ACM Computing Surveys*, 24:63–113, 1992.

[16] Hyunjung Park, Richard Pang, Aditya Parameswaran, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. An overview of the deco system: Data model and query language; query processing and optimization. *ACM SIGMOD Record*, 41(4), December 2012.

[17] Hyunjung Park, Aditya Parameswaran, and Jennifer Widom. Query processing over crowdsourced data. Technical report, Stanford University, September 2012.

[18] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[19] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, It. A. Lorie, and T. G. Price. Access path selection in a relational database management system. pages 23–34, 1979.

[20] Beth Trushkowsky, Tim Kraska, Michael J. Franklin, and Purnamrita Sarkar. Getting it all from the crowd. *CoRR*, abs/1202.2335, 2012.

[21] Beth Trushkowsky, Tim Kraska, Michael J. Franklin, and Purnamrita Sarkar. Crowdsourced enumeration queries. In Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou, editors, *ICDE*, pages 673–684. IEEE Computer Society, 2013.

[22] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Crowder: crowdsourcing entity resolution. *Proc. VLDB Endow.*, 5(11):1483–1494, July 2012.

[23] Jiannan Wang, Guoliang Li, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Leveraging transitive relations for crowdsourced joins. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *SIGMOD Conference*, pages 229–240. ACM, 2013.

[24] Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. Question selection for crowd entity resolution. Technical report, Stanford University, 2012.