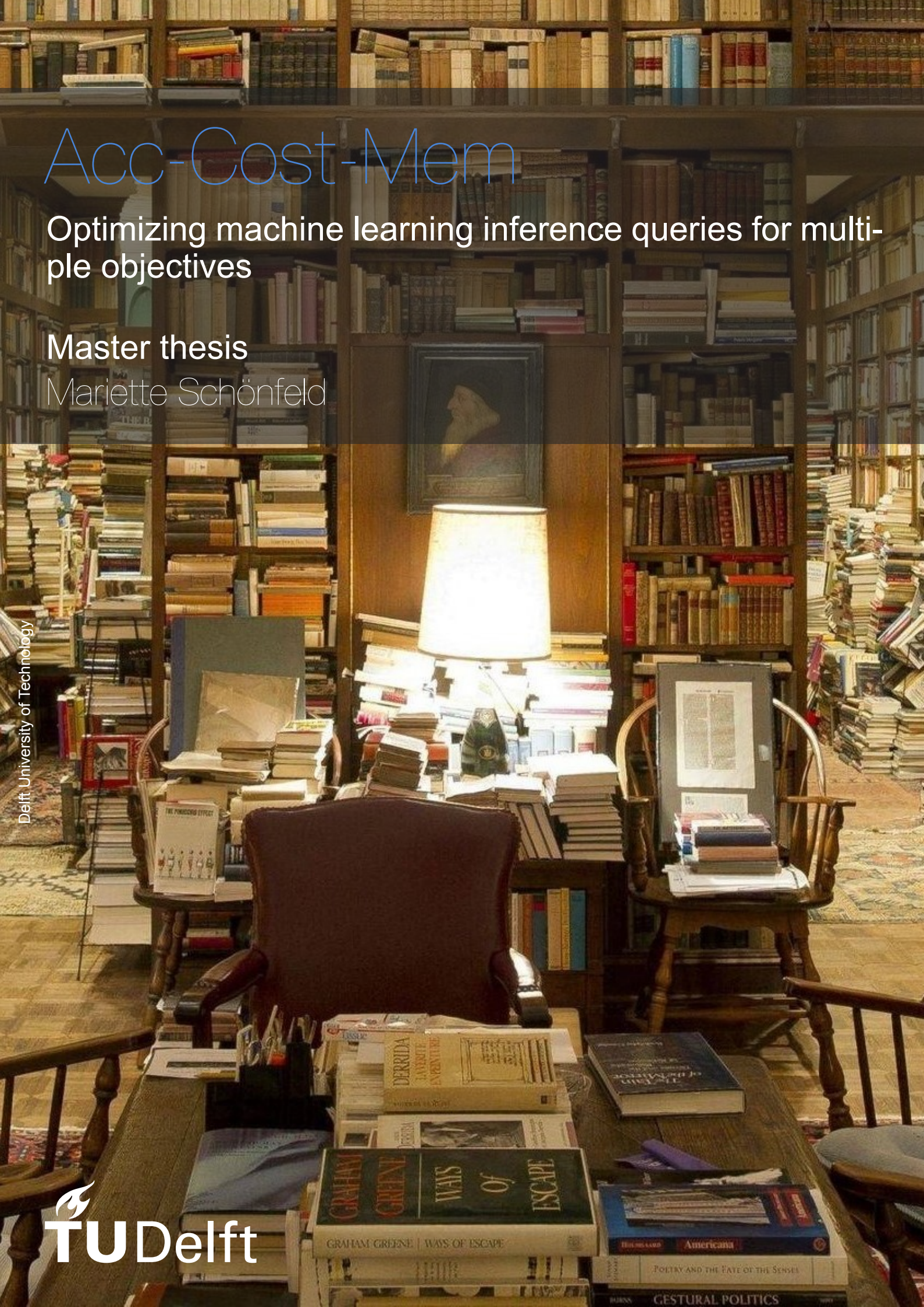


Acc-Cost-Mem

Optimizing machine learning inference queries for multiple objectives

Master thesis

Mariette Schönfeld

Delft University of Technology

TU Delft

Acc-Cost-Mem

Optimizing machine learning inference queries for multiple objectives

by

Mariette Schönfeld

Instructors:	G. Houben, A. Katsifodimos, R. Hai
Teaching Assistant:	Z. Li
Project Duration:	February, 2022 - December, 2022
Faculty:	Faculty of Electrical engineering, Mathematics, and Computer Sciences, Delft

TU Delft

Preface

This work concludes 7.5 years of being a student at TU Delft (for now). I did not know what to expect from doing a thesis, but this was certainly not it. First of all the topic was supposed to be in Robotics, but research takes you in interesting directions. Second, I did not expect to enjoy it so much, but also find it so hard simultaneously. I had a tough year with every month a new illness to deal with, in addition to suffering from chronic illness all the while. I'm proud what I have achieved under these circumstances.

This work would not have been possible without the help of others. First of all my supervisors from TU Delft. Asterios, you have been wonderful from the first time we met, and your sense of humour and disdain for formalities always managed to make me leave our meetings happier than I came in. Your enthusiasm for all the directions I wanted to explore really elevated my work. Rihan, I rarely have met someone with so much rigor and attention to detail, and your feedback always helped me move forward from whatever slump I was in. Last but certainly not least, my sincere thanks to Ziyu. I'm sure that supervising a chaotic person like me was difficult at times, but you never once complained about all the texts I sent about things I didn't understand. I'm really excited to see how the end of your PhD will turn out. You three have really changed my views on working in academia, and I hope to be pursuing a PhD after this project. Being able to call you my supervisors is truly an honor.

Second, my supervisors from Cognizant. Hayo, your experience in working with students of all backgrounds was really pleasant in the times I was stressed. You always say that you lack the technical background, but there were so many times where I needed general project-guidance, or tips on working with busy people, and you truly stepped up. Vijay, thank you for all your calm sobriety during our weekly meetings. In addition to my supervisors, I also need to thank Chris Molanus from Cognizant. Your excitement for academia as a non-academic always made me excited to catch up during the bi-weekly Data Science Guild Meetings, and giving me access to your private cluster really helped me make some needed last-minute changes to my experiments.

Last, my friends, family, all the wonderful people at Krashna Musika, and my fellow board members there. It is probably customary to leave it at that, but so many people have pulled me through difficult times and not naming them personally would be reductive in my opinion. First my roommates: Anouk, Ilja, and Sanne. I could always count on late-night tea to vent about my difficulties, and going out to help me let things go. Second, Debora: for months we worked tirelessly on our theses, and I'm so proud that we are now finally both actual engineers. Julia, you have moved abroad but I can always be sure of your support. Olmar, you are a good colleague and a better friend. Lastly, my biggest fan (as far as you can have fans in academics): my mother.

Mariette Schönfeld
Delft, December 2022

Summary

Machine learning inference queries are a type of database query for databases where a model pipeline is needed to evaluate its boolean predicates. Using a model zoo it is possible to select a variety of models to execute in a sequence rather than using a highly specialized model to answer every query predicate. Machine learning models can have multiple measurements for gauging performance however, and the quality of a query plan therefore is not only dependent on the time needed to compute it. Selecting a query plan of models that balances multiple objectives is not a trivial feat however.
This work builds upon existing methods that utilize MIPs for model selection and ordering for machine learning inference queries by extending them with multi-objective optimizing capabilities. The opportunity for adding a third objective, namely memory footprint, to that of accuracy and execution cost is explored. Several methods are then considered and compared on their suitability, and the final chosen method, the Archimedean goal method, can generate Pareto optimal query plans that provide gains over naive, greedy methods. In addition, several methods of cutting down runtime on the original optimizer are explored, leading to a program than can generate higher quality solutions in less time.

Contents

1

Introduction

During this process one aspect the author of this thesis found notably difficult about this topic was the bridge between explaining the topic understandably to anyone, versus the technical parts of it that make it challenging to solve. The problem sounds fairly easy once explained in layman's terms, and through back-and-forth discussion it is revealed that the topic is more complex than initially thought. Because such a discussion where one party is constantly subverted by the other seems to be the way to explain this problem, we have therefore decided to practice our creative writing skills and write such a discussion as our introduction. The goal is that most readers can understand this section, and that formal definitions follow in the next chapters.

Consider two police officers, one a detective and one a younger constable. They have just heard that a hit-and-run has been done, and they are tasked with finding the culprit. The witness mentioned that the culprit was in a **yellow Volkswagen Polo**, but a number plate was not remembered. As the priority to find this person is extremely high, there is no time to find a vehicle registry. Luckily the country they live in has cameras on all roads, and due to the remarkable appearance of the car, they want to attempt to find the car using real-time monitoring of the roads.

They quickly realize that monitoring all cameras is way too much data at once, and therefore the constable is tasked with finding a solution that filters the data efficiently. After a quick google search he comes back: ”Detective, I have a great idea to solve our problem! We can use artificial intelligence to identify cars on the road.” ”Great work constable. How does AI work?” replies the Detective. ”It is easy, ” says the constable, ”we just plug in the images we take from the live-cameras into a so-called **machine learning model**, and the model tells you which car is in the image.” ”Wonderful! Have we deployed it yet?” ”No, there is a small issue. I couldn't find a model that classifies both color and car type. To make a model do that, we need the right data, a lot of time, and expensive hardware. We have none of that.” ”Ah, but maybe we can find two models to do that for us? Then we can run a model that recognizes car types, and afterwards an model that classifies yellow objects.” ”What a great idea detective! I will look for such models.”

A few minutes later the constable returns: ”Good news detective! The method you suggested has already been researched, it is known as a **machine learning inference query**, it is like a database query, but where machine learning models are needed to evaluate the data. The formal query in our case is

$$q := yellow \ \wedge VWPolo$$

where the wedge operator $\wedge$ signifies ”and” and the vee operator $\vee$ signifies ”or”. I have found several models that suit our needs, I have put them in table. The combination of a list of models and such a table is apparently called a **model zoo**.

Model name	**Cost**	*yellow*	*VW Polo*
model 1	10	80%	
model 2	30	95%	
model 3	20		85%
model 4	40		95%

”For every model I have denoted the accuracy and the amount of milliseconds it takes to run on a single image, which we also call cost.” ”Great work constable! What do you mean with accuracy however?” ”Well, the models are not perfect. We expect them to get it wrong in some cases, so model 1 is right about *yellow* in 80% of cases.” ”Noted, so let’s just use the most accurate two models.” ”I was afraid you would say that, but models 2 and 4 together take too long to be used in busy traffic. Once the algorithm has finished, another car has passed undetected.” ”That is not ideal constable. So we would want to select two models, that have somewhat high accuracy, but where they are still quick enough. What are our options?” ”I have taken the liberty to draw up a table for our options, with the composite accuracy and total cost for the models”:

Query plan	**Accuracy**	**Cost**
model 1 & 3	68%	30
model 1 & 4	76%	50
model 2 & 3	81%	50
model 2 & 4	90%	70

”Interesting. The accuracy and cost of these query plans differ quite drastically. The only solution I think we shouldn’t choose is model 1&4: it has the same cost as when we choose model 2&3, but lower accuracy.” ”Agreed Detective, but then we are still left with three solutions. How do we choose one?” ”I suppose we need to first think about whether accuracy or cost is more important, and how much. Then based on those preferences, I guess we also need to find a method of determining our best candidate”. The two men take a few minutes to think about methods, and resume their conversation after.

”Shall we calculate the maximum amount of time we have, and then choose the most accurate combination that is quicker than that time?” ”That is a good idea Detective, but I’m not sure if we can define such a bound. The cars drive at different speeds on different roads.” ”Good point. Maybe we can first find the highest accuracy possible, and then see if there are combinations with the same accuracy that are quicker.” ”Also a good idea, but then we prioritize accuracy over cost disproportionally.” ”Maybe we can combine our preferences in a way that we can calculate a score for every model, and choose the highest score?” ”Such a utility score would be great! But then the way we define this utility function has a lot of bearing on which combination we will find.” ”Exactly detective, there are a lot of possibilities and there is not a solution that is the ’best’. Balancing accuracy and cost effectively is quite complex frankly, it’s a good thing that there are only 4 combinations that we have to consider.”

At this precise moment, the officers receive a call: the witness that saw a yellow Volkswagen Polo actually has amended their statement. Apparently they are blue-yellow colorblind, and have very poor knowledge of cars. The car could also have been a VW Golf. ”Let’s immediately amend our machine learning inference query, to reflect that the car could have been blue or yellow, and must be a Polo or a Golf:

$$q := (yellow \vee blue) \wedge (VWPolo \vee VWGolf)$$

”Constable, please go look for more models.” ”Already on it detective”. The constable returns with the new model zoo:

Model name	Cost	*yellow*	*blue*	*VW Polo*	*VW Golf*
model 1	10	80%			
model 2	30	95%			
model 3	20			85%	
model 4	40			95%	
model 5	10		80%		
model 6	25	90%	90%		
model 7	20				85%
model 8	35			90%	94%

”I have also managed to find two models that can classify two predicates at once” ”Great idea constable, if we use model 6, we can classify two colors at once and be quicker. But this is getting quite complicated, how many combinations are possible now?” ”Already 36 detective.” ”Oh god, that is already a lot more than when we had two predicates. And I just realized, that if we first check for yellow and blue, which are rare colors for cars, we might not have to check for the car type in most cases, as we have already excluded the car if they are not of these colors. So the order in which we execute the models is also important.” ”Aha, so we need to find a selection of models and order to execute them in, that answers our query but balances accuracy and cost, based on our needs. I will go back to see if there has been research on balancing accuracy and cost for machine learning inference queries using model zoos.”

Which is where our story ends, as this has not yet been researched in great detail and is therefore the topic of this thesis. By accident, the detective and the constable found an open problem in academic research. They were able to formulate their problem, namely finding cars on the road of a specific color and type, as a machine learning (ML) inference query. They didn't have the resources to train their own model, but luckily stumbled upon a model zoo that contained models for their needs. They quickly found out that finding a combination of models becomes quite difficult when the amount of query predicates increases, especially when you want to effectively juggle two objectives at once. They also realized that multi-objective optimization (MOO) is a 'fuzzy' problem: the quality of a solution is defined by the needs of the user. Given the fact that the performance of ML models can be described in many more ways than accuracy and cost, the real life version of this problem becomes even more complicated. To summarize, we are looking for a method to find query plans (a selection of models and an order in which to execute them) that balance multiple objectives (accuracy and cost) for a machine learning inference query (a database query where a model pipeline is needed for its evaluation) when a model zoo (a collection of models) is available for our use.

In this work we therefore investigate current methods for finding query plans for ML inference queries and see if they can be extended to have multi-objective optimization capabilities. For the optimizer that we found, we implement a collection of methods that juggle objectives, and investigate whether more objectives can be added to it. Along the way we propose some methods of improving the original optimizer for computational complexity and solution quality. Lastly we test the performance of our optimizer on actual data using real models, and see if it juggles objectives better than naive methods. The application domain this work has chosen for its practical experiments is that of natural language processing (NLP). Language is naturally very multi-faceted, and meanings of sentences can change due to the placement of a comma. This nuance lends itself particularly well for the ml inference query approach of looking at data. Additionally NLP is known for its models needing expensive resources for efficient training, and it is much more normalized to use pretrained models [33].

1.1. Research contributions

To summarize, this work will concern itself with optimizing machine learning inference queries for multiple objectives using model zoos. The main question therefore is:

How do you derive a query plan for a machine learning inference query that balances multiple objectives using model zoos?

This question can be divided into several sub-questions:

- What is the current standard for deriving machine learning inference query plans?
- How do you define an optimizer for such a query that takes multiple objectives into account?
- Is there opportunity for adding other objectives?
- How does this optimizer perform against naive methods?
- How does this optimizer perform on NLP tasks?

1.2. Thesis structure

This thesis will be following the below structure:

Problem formulation which contains a formal description on the ML inference query problem and the current state of the art. A small introduction to Machine Learning is given.

Background which will discuss theoretical background on query- and MO optimization.

Optimizer proposal wherein a new optimizer is introduced, including new objectives. The optimizer's performance is then compared with its competitors.

Experiments the model is then used to generate query plans for some NLP tasks. The query plans are then executed on a dataset in order to compare the query optimization approach versus traditional methods.

2

Problem definition

As mentioned before, we aim to generate efficient query plans for ML inference queries when a model zoo is available. This chapter will concern itself with formulating some definitions and formalizing the problem. To summarize very briefly, we discuss theory on the following topics:

Machine learning models which we use in our machine learning inference queries to estimate the value of predicates.

Model zoos that are databases of ML models, where information of their performance is also stored.

Machine learning inference queries which are database queries that query on databases such as images. We need a different type of query because we need a machine learning model to evaluate truth of a predicate.

Query plan which is a set of models from the model zoo to answer the query on data. We want to optimize this query plan to be both accurate and quick.

Lastly we discuss some related work to show where current research is headed on similar topics, demonstrating that our work is both novel and relevant.

2.1. A (very) short introdution to Machine Learning

A large portion of this work assumes some background on machine learning. However, not enough knowledge is needed to dedicate a full section to it in the 'Background' chapter. This chapter might be somewhat difficult to understand however without a small refresher on the basic terminology.

A machine learning model is an algorithm that learns to make decisions based on data [4]. For this problem we limit ourselves solely to binary classification problems: a data point either satisfies a certain predicate or it doesn't. ML models are 'trained': they look at a lot of data that is correctly labelled, and attempt to make a set of rules that allows them to correctly classify previously unseen data. ML models typically do not have perfect accuracy: they classify some data points incorrectly. Therefore the accuracy of a model is an important measurement. There are multiple metrics to represent accuracy, but they typically model a certain percentage, namely how good the model is at predicting data. The most general way of doing this is dividing the amount of correctly classified data points by the total amount of data points.

For this work very little understanding of actual machine learning is needed, other than that a ML model is a (black box) function that takes as input a data point and classifies a certain predicate as true or false, with a certain accuracy. For further reading on machine learning the reader is referred to one of the most exemplary books in the field, "Pattern recognition and Machine Learning" by Christopher M. Bishop [4].

2.2. Model zoos

A model zoo is a database where multiple ML models are stored, including meta information such as their execution cost, or accuracies on certain tasks. Examples of public model zoo include Hugging-

Face [15] and the TensorFlow Hub [29]. The idea is that users can use these models without having to train, and still achieve good accuracy.

We define a model zoo as $R = (M, I, P, A, C)$: a tuple of a set of models M, a set of classes I the models in M can inference on, the set of accompanying boolean predicates P, an accuracy matrix A and a cost matrix C. A model zoo accompanies datapoints Ξ where for $0 \leq k \leq |M| - 1$ a model $m \in M$ is defined as a function

$$m_p : \Xi \to \{0, 1\}$$

that classifies the corresponding class $i \in I$ using a binary value (e.g. "does an image $\xi \in \Xi$ satisfy p according to model m?"). M, I, P are unstructured sets, but $A \in [0, 1]^{|M|}$ and $C \in \mathbb{N}^{|M|}$ are matrices of size $|M| \times |P|$ and $|M| \times 1$. A serves as a lookup table for the accuracy per model on a given predicate, and C as a lookup table for the execution cost of a model. The execution cost is measured in milliseconds, which of course varies per hardware setup and therefore serves as more of a relative measure.

For the purposes of future legibility, we are elucidating the indexing methods further. Consider a 'dummy' model zoo:

Model name	**Model index**	**Cost**	*sentiment*	*person*	*object*
LR	0	5	0.9	0	0
SVM	1	10	0.95	0	0
DNN1	2	20	0	0.92	0.93
DNN2	3	25	0	0.95	0.97
DNN3	4	15	0	0.98	0
DNN4	5	15	0	0	0.99

Table 2.1: A (dummy) model zoo for determining the mentioning of objects and persons in a text, in addition to the sentiment regarding them.

This dummy zoo contains six models and three classes, *sentiment*, *person*, and *object*. *Sentiment* would be the task of classifying a text as having either positive (1) or negative(0) sentiment, and *person* and *object* as the text containing mention of a person and/or object respectively. In this case,

$$A = \begin{pmatrix} 0.9 & 0 & 0 \\ 0.95 & 0 & 0 \\ 0 & 0.92 & 0.93 \\ 0 & 0.95 & 0.97 \\ 0 & 0.98 & 0 \\ 0 & 0 & 0.99 \end{pmatrix}, \text{ and } C = \begin{pmatrix} 5 \\ 10 \\ 20 \\ 30 \\ 15 \\ 15 \end{pmatrix} \tag{2.1}$$

and $M = \{$LR, SVM, DNN1, DNN2, DNN3, DNN4$\}$ and $P = \{$sentiment, person, object$\}$. We derive information from the model zoo similar to array indexing. In general $A_{m,p}$ refers to the accuracy that model $m \in M$ has on predicate $p \in P$, and C_m the projected execution cost. For example, we use $A_{\text{SVM, sentiment}}$ to denote the accuracy of the SVM model on sentiment classification, which in this case is 95%. When using this model, the expected execution cost C_{SVM} is then 10 milliseconds. The SVM model has 0% accuracy on classifying *person* and *object*, which is shorthand for meaning that the model is not trained for classifying these predicates, and preferably should not be used to do so.

2.3. ML inference queries

The concept of ML inference queries is somewhat novel, with the first mention in a 2019 paper [19]. This paper defines an ML inference query as "a SQL query that invokes a model pipeline". Other works that tackle the problem agree on this semantic definition [21][23], but do not have consensus on a strict mathematical definition that is rooted in relational algebra [6]. This work does not claim to have a definition to solve this problem, but puts forward one that makes sense for this work specifically. For other definitions the reader is referred to the works of Li et al. [21] and Lu et al. [23].

Consider a set Ξ of data points of the same type (e.g. a set of images, or an NLP corpus.) Let $R = (M, I, P, A, C)$ be an accompanying model zoo. We define a machine learning inference query on the set of predicates as a database query on Ξ that uses relational operators $\{\vee, \wedge\}$ and evaluates a predicate $p \in P$, $0 \leq k \leq K$ using a model $m_p \in M$ on a data point $\xi \in \Xi$. In this work we focus on conjunctive normal form (CNF) and disjunctive normal form (DNF) queries. A CNF query is a disjunction of separate conjunctions:

$$q_1 := (p_1 \vee p_2 \vee ... \vee p_k) \wedge (p_{k+1} \vee ... \vee p_{k+i}) \wedge ... \wedge (p_n \vee ... \vee p_{n+j}) \tag{2.2}$$

and a DNF query is a conjunction of separate disjunctions:

$$q_2 := (p_1 \wedge p_2 \wedge ... \wedge p_k) \vee (p_{k+1} \wedge ... \wedge p_{k+i}) \vee ... \vee (p_n \wedge ... \wedge p_{n+j}) \tag{2.3}$$

These queries contain the same set of predicates ($\{p_1, p_2, ..., p_{n+j}\}$ but differ in the way that they are conjoined. For q_1 $(p_1 \vee p_2 \vee ... \vee p_k)$,...,$(p_n \vee ... \vee p_{n+j})$ form the predicate groups, while for q_2 these are $(p_1 \wedge p_2 \wedge ... \wedge p_k)$,...,$(p_n \wedge ... \wedge p_{n+j})$. Predicates can be answered by any model from R (but preferably, $A_{m,p} > 0$), and models can be used for evaluating multiple predicates.

In formal query optimization jargon, the models M are known as ”user defined functions” (UDFs) and they tend to be expensive to execute, and hard to optimize for. Finding a good query plan is therefore both difficult and important.

2.4. Query plans and optimization

The traditional definition for a query plan is ”a sequence of steps used to access data in a SQL relational database management system”. In this work we are not working with traditional relational database management systems, but with model zoos and datasets. The semantic definition still stands however: based on a query, a query plan is a sequence of steps to complete in order to extract the data points from the dataset that fit with the query.

Therefore for ML inference queries a query plan consists of a set of models $\{m_{\alpha_1}, m_{\alpha_2}, ..., m_{\alpha_{n+j}}\}$ from R and an order in which they should be executed on data points $X \subset \Xi$. A formal definition for evaluating an ML inference query using q_1 on a data point $\xi \in \Xi$ is then:

$$(m_{\alpha_1,p_1}(\xi) \vee m_{\alpha_2,p_2}(\xi) \vee ... \vee m_{\alpha_k,p_k}(\xi)) \wedge \tag{2.4}$$
$$(m_{\alpha_{k+1},p_{k+1}}(\xi) \vee ... \vee m_{\alpha_{k+i},p_{k+i}}(\xi)) \wedge ... \tag{2.5}$$
$$\wedge (m_{\alpha_n,p_n}(\xi) \vee ... \vee m_{\alpha_{n+j},p_{n+j}}(\xi)) \tag{2.6}$$

Because the models from M return boolean values it is then possible to discard data points for further inferencing based on intermediate evaluation. If we consider the query from the introduction:

$$q := (yellow) \wedge (VWPolo) \tag{2.7}$$

We can first run an ML model that classifies $yellow$. All the data points that return 0/false then do not have to be considered anymore by the model that checks $VWPolo$. In other words, we only continue data points down the pipeline for which there is the possibility that the query is true. This means that the models down the pipeline inference on smaller amounts of data. It should be apparent that the order of execution of different models can influence the total execution time, depending on how selective a predicate is.

Typical query optimization concerns itself with finding a good sequence of steps that finds all the data in a relational data model [6] that fits your query quickly. In this case, the data that is returned is also dependent on which models are chosen for answering the query: if a model with bad accuracy is chosen in the query plan, perhaps many data points that fit the predicate are removed for further evaluation because of a false negative (and datapoints that do not actually fit the predicate are kept in). Therefore total time for filtering the dataset is not the only measure of performance, but also how many data points get correctly classified.

2.5. ML inference query optimization

The work presented by Li [21] formulates the ml inference query optimizer as a non-linear mixed integer program that either maximizes accuracy or minimizes cost while bounding the other objective. The idea of the research questions for this thesis follows Li's work very closely, and therefore this work's proposed optimizer will be based on Li's work.

As mentioned before in the introduction, both accuracy and cost (and other possible objectives) are important to an ML inference query query plan. This makes it a "multi-objective optimization problem": more than one metric needs to be optimized in order to consider a solution as good. In fact, there are many more metrics to be considered when it comes to evaluating the performance of a machine learning model. These metrics could therefore also be relevant to ml inference query optimization, as the metrics of individual models also influence the total performance of the query evaluation.

Therefore we would like an ML inference query optimizer to take both those objectives into account, but also allow for the addition of other objectives in a simple way.

2.6. Related work

2.6.1. ML inference queries

As this work closely follows Li [21]'s optimizer, the details of this work will be discussed in another chapter. Other works that investigate ML inference queries include Probabilistic Predicates (PPs) [23] and Tahoma [2]. PPs attempt to prevent expensive ML models from being run for machine learning inference queries by starting with cheap binary classifiers. This work is similar to Li's but not necessarily competitive: the methods can work together on a query. Tahoma does not necessarily specialize itself for ML inference queries, but the method proposed also concerns itself with pushing more expensive models down the pipeline and first executing the simpler ones is similar to what we are trying to achieve. To the author's knowledge there isn't yet work that optimizes ML inference queries using multiple objectives in a sophisticated way.

Other work that is similar to ML inference queries but a little bit farther removed from our intended approach is BlazeIt [17] and NoScope [18]. BlazeIt attempt to speed up video processing by utilizing spatio-temporal information. It treats the video as a database to be queried upon. NoScope is quite similar in that it utilizes spatio-temporal information for classifying video quickly, but trains a cascade of models that are specialized for the target on the fly.

2.6.2. Mixed integer (linear) programming in query optimization

As mentioned before, we want to have an optimizer that easily allows for addition of objectives and can extend to MOO capabilities. The optimizer proposed by Li [21] is a good starting point, and they describe a Mixed integer program (MIP). This is not the only case of using MIPs/MILPs for query optimization however. In 2016 Immanuel Trummer and Christoph Koch wrote a paper that solves the join problem using an MIP [31]. The results are impressive, and their optimizer handles much larger input than usual with ease. A similar paper written in 2007 by Stratos Papadomanolakis and Anastassia Ailamaki uses an ILP to tackle the indexing problem [26]. The result is an optimizer that again scales well with large input and receives all the benefits of classical programming.

2.6.3. Multi-objective query optimization

Like mentioned previously, multi-objective query optimization lends itself particularly well for ML inference query optimization because of the amount of metrics that can be used to describe an ML model. There is however some research into multi-objective query optimization, using objectives such as the amount of intermediate memory that is needed for the pipeline in addition to execution cost. A 2016 paper by Florian Helff, Le Gruenwald and Laurent d'Orazio uses the weighted sum method for optimizing mobile-cloud database environments. The work scores query plans using the weighted sum method, starting off from a lexicographical approach. Another paper by Immanuel Trummer and Christoph Koch [30] attempts to generate the set of Pareto-optimal query plans.

2.7. Summary

After formally introducing several definitions, the total problem can now be formally identified. Given a model zoo $R = (M, I, P, A, C)$ and a machine learning inference query q in either CNF or DNF form, find a query plan $\mu \subseteq M$ that is optimized by taking multiple objectives like accuracy and execution cost into account. Preferably the optimization method also allows for adding other objectives easily. Before we continue with the background topics of this work (query optimization and traditional optimization).

3

Background

In the previous chapter we discussed some theoretical frameworks. The goal is to find good query plans (i.e. selection and ordering of ML models) to extract data that satisfies a machine learning inference query, when we have a model zoo available (i.e. a collection of models). The difficulty is that the models perform differently, and that there might be multiple models that can classify the same predicate. Therefore there are multiple query plans available, that have different execution costs and accuracies. The model proposed by Li et al. [21] provides a solution to the problem, but does not use the most sophisticated method to balance accuracy and cost. Therefore we explore in this work which methods are more appropriate.
The following chapter discusses some background around optimization and query optimization. To summarize the sections:

Traditional query optimization: finding a query plan is a query optimization problem. In our case however, traditional query optimization techniques are not adequate, which we demonstrate in this section. We also discuss why traditional optimization is a better choice for this problem.

Single-objective/traditional optimization: as MOO is a variation on single objective optimization, we discuss optimization in general.

Multi-objective optimization: after establishing some background on optimization, we can discuss the theory and terminology on MOO.

3.1. Traditional query optimization

Like mentioned before in section 2.4, finding a good query plan for machine learning inference queries is a query optimization problem: a certain amount of data that fits a query needs to be selected from a dataset, and we would like to do this accurately and efficiently. Most research regarding query optimization concerns the traditional relational database model [6], where data is represented as a set of tuples. For ML inference queries, the data is of a different identity: the datasets that ML models typically solve tasks for are unstructured and contain datatypes that can't be easily represented as a tuple, and require ML model inferencing in order to derive the content. Images for example can be represented as a tuple of pixel values, but individual pixel values do not provide a lot of information on the content of the image.

This is part of the reason why traditional query optimization approaches fail for ML inference queries, and therefore a more specialized optimizer is needed. In order to further demonstrate this need, we discuss a few simple query optimization heuristics [8] and why they are not sufficient. The heuristics we consider are:

1. Do selections as early as possible
2. Do projections as early as possible
3. Avoid Cartesian products

For each of these rules we will briefly discuss why this is not generally applicable to machine learning inference queries. The point is not to illustrate that no traditional query optimization technique cannot

be used, but rather that the most basic rules are not applicable and more sophisticated methods are necessary.

3.1.1. Selections

A selection is a database operation where all data is filtered on a certain rule (e.g. select all female citizens from a population register). The general goal of this rule is to save time by only moving information down the pipeline of which you are suspecting that it fits your query. The problem with machine learning inference queries that predicates first need to be evaluated by a machine learning model before a selection can be applied. The boolean predicate upon which selection is applied (e.g. $female = true$ in case of the population register) can only be provided upon execution of an ML model, an expensive operation. Rather than moving down selections, choosing which models are cost-efficient to execute first is a more important question. This is more an issue of balancing the accuracy and the cost of the model, also in comparison to the models you are picking for other predicates.

3.1.2. Projections

A projection is a database operation where you discard certain information of data points which you don't expect to use for this particular query. For example, in the case of filtering a population register on citizens that have a certain age and gender, you can discard their social security number as it is not relevant to this query. This way you are only continuing with strictly necessary information down the pipeline. For machine learning inference queries, the entire data point needs to be fed into a machine learning algorithm. In case of an image database, the picture needs to be used for input in its entirety. Compressing the image is of course possible, but you would have to use algorithms that likely use lower resolution, and this compression would hold for the entire query evaluation, influencing the rest of the model 'stack' as well.

3.1.3. Cartesian products

A Cartesian product is a method of combining two tables. For example in the case of the population register, if you would want to find the citizens that have a double passport, you could combine the tables of different countries and check which entries have the same name (practicality of this scenario aside). This is typically a very expensive operation, and therefore you would want to do this as late as possible.

The datasets used for machine learning are generally more like unstructured sets, rather than a collection of tables that relate to each other like relational databases. Therefore there is generally no need to use a Cartesian product (or any type of join).

3.1.4. The case for traditional optimization

To summarize, these basic query optimization techniques either do not work or are irrelevant to this particular problem. It is possible however to optimize queries in different ways. This problem closely resembles traditional assignment/selection problems. A selection/assignment problem is an optimization problem where a set of variables needs to be matched, or a selection needs to be made. A typical selection problem is the facility location and assignment problem. In this problem parties need to be assigned to facilities with minimal cost. But if there are multiple locations where facilities can be opened (and a facility costs money to open), but the cost of a party being assigned to a facility is also distance-dependent, how do you select where to place facilities, and which parties do you assign where?

The ML inference query optimization problem can also be translated to a similar problem. You namely select the right model, select their execution order, and their assignment to predicates. But rather than minimizing cost we also need to optimize for accuracy. In mathematics, (mixed integer (linear)) programming (MIP/MILP) is used oftentimes for selection and assignment problems, and is therefore a sensible choice for this problem. As there is ample research of multi-objective optimization for MIPs, extending Li's model is a sensible choice.

3.2. (Single-objective) optimization

Optimization is the mathematical field of finding a solution in a broad solution space where the utility of solutions is evaluated using some sort of objective function, referred to as cost. Typically, the idea is

to find the solution that has minimum cost in the space. Finding the solution with maximum utility is the same process, and therefore the term 'cost' is also used for maximization problems. Depending on the size and shape of the solution space, finding the optimal solution can be time and resource intensive if not impossible. A method that concerns itself with finding a somewhat good solution in better time is called a heuristic.

Linear programming is a branch of optimization where the solution space is a subset of $\mathbb{R}^n_{\geq 0}$ that can be represented by a finite set of linear equations, thereby making a convex polytope. The canonical form for this time of problem is

$$\max c^T x \tag{3.1}$$
$$\text{such that } Ax \leq b, \tag{3.2}$$
$$x \in \mathbb{R}^n_{\geq 0} \tag{3.3}$$

Where c is a vector in $\mathbb{R}^n$ that denotes the cost, and $A, b \in \mathbb{R}^n$ a matrix and vector respectively that span the solution space. x is oftentimes referred to as the decision variables. In many cases, it can be more intuitive to represent the solution space as a finite set of linear equations rather than a matrix multiplication. This canonical form always describes a convex polyhedron, and there are three scenarios regarding optimality:

infeasibility when one or more inequalities make the solution space empty and no solution exists.

unboundedness when the convex polyhedron is unbounded, and therefore the optimal solution is unbounded

bounded and feasible when the convex polyhedron is both feasible and bounded (therefore a convex polytope), there exists a set of optimal solutions that can be described by a finite set of equations.

Even more striking in case of a bounded and feasible polyhedron, the optimal solution can be found on one or more (when multiple solutions return the same objective value) of the polyhedron's vertices and can be represented as a linear combination of these points. This is a very powerful theorem, and forms the basis of optimization algorithms. In short, as long as you can formulate your optimization problem as a set of linear equations, finding the solution is as simple as efficiently pivoting over the vertices.

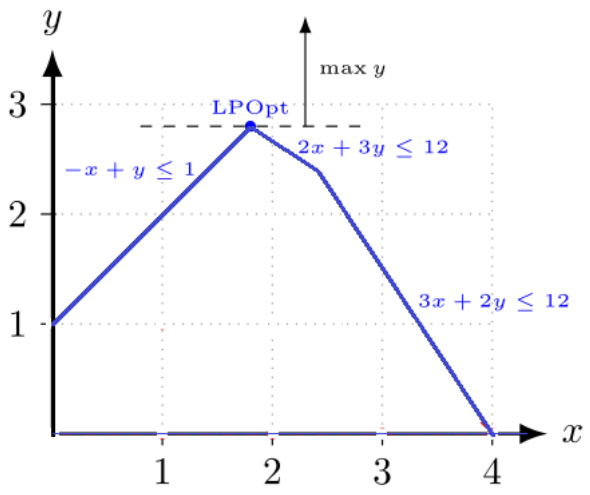


Figure 3.1: A convex polytope spanned by a set of linear equations. Note how the optimal solution is located on a vertex of this polytope.

3.2.1. Integer optimization

A critical assumption for linear programs (LPs) is the continuity of the solution space. In many optimization problems however, solutions need to be (partially) integral (e.g. in assignment or selection problems). In this case, the solution space is still a convex polyhedron, but the vertices are oftentimes not located on integer points. Finding the optimal solution is sadly much more complicated than rounding the optimal non-integral solution down, and obtaining the convex hull (set of integral solutions in

the convex polytope) is also difficult. In short, the criterion of integrality often adds a lot of difficulty to optimization problems. In particular, integer programming is NP-hard while linear programming is in P.

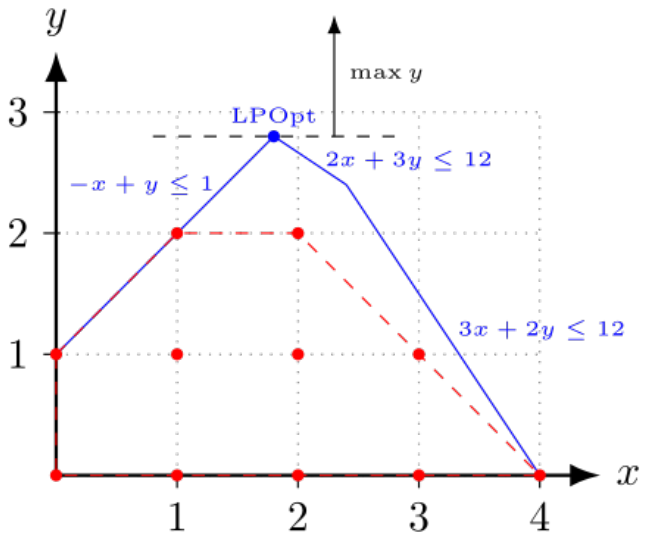


Figure 3.2: An ILP. Note that the LP solution is located on a non-integer point. The convex hull is spanned by the red dots.

An interesting corollary is that solving an ILP is equivalent to finding the convex hull of the LP relaxation and solving this as an LP. Finding the convex hull of a convex polytope however is also not an easy feat. Generally ILP solving methods use algorithms that use a starting solution, branch out to alternatives, and prune solutions which cannot improve on the current optimal solution.

3.2.2. Nonlinear/quadratic optimization

As mentioned before, as long as one can formulate their solution space as a finite set of linear equations in a continuous space, optimization can be a manageable problem. Sadly, integrality is not our only enemy. In many applications the solution space can not be described by linear equations, and finding the optimal solution is not as simple as locating the vertices on a polyhedron or using branch-and-bound. For this work we only provide background on non-linear polynomial programming. Non-linearities in the equations spanning the solution space can lead to a non-convex search space. In layman's terms, a convex space is a space where you can draw a straight line between any two points in the space, and the line is contained entirely in the space. A non-convex search space is harder to navigate for solvers.

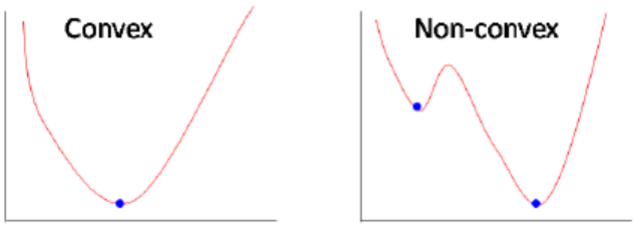


Figure 3.3: A simple graph illustrating the differences between a convex and a non-convex space

To quote Tyrrel Rockafeller in Siam review 1993: [28]:

> "...in fact, the great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity."

Without going into details about the theory surrounding convex versus non-convex optimization, a simple analogy to illustrate why non-convex optimization is more difficult can be used. Imagine a person

searching for an object in a non-furnished room. If the room is rectangular, the person could theoretically see the entire room from wherever they are standing. If the room has an L-shape, the person would have to move around to be able to see the entire space. Even more problematic is the fact that the person does not know where the room ends, unless they walk around the corner. For all they know the room might be infinitely long around the corner. In other words, the difficulty of searching a non-convex space is somewhat analogous of not being able to look around corners.

In many cases, product variables can be avoided using linearization methods that utilize knowledge of the solution space [3]. Therefore, to avoid non-convexity, it is considered good practice to avoid product variables when possible.

3.2.3. Commercial optimizers

In short, integrality and non-linearities/non-convexity are the enemies of optimization. These types of optimization problems need potent solving methods, especially when reaching a certain amount of variables and/or equations. From now on we will not reason about the inner workings of solving algorithms and how they navigate search spaces, but keep in mind the problems that can lead to a difficult search to navigate. In other words, we treat the solving algorithm as a black box.

The sophisticated solvers are generally not open source (though some offer academic licenses) and compete in terms of solving speed and quality of solutions [1].

For this specific work one solver was used, namely Gurobi [12]. Originally presented in 2009, the solver is regularly updated with new features. The main considerations for using this solver over others in particular:

1. The solver boasts competitive benchmarks to other solvers [1].
2. Gurobi provides academic licenses with a simple procedure.
3. Gurobi supports non-linear integer optimization since 2020 as of version 9 [14].
4. Gurobi provides extensive customer support with a user forum.
5. Gurobi has a sophisticated Python platform.

This combination of merits was not found in another solver to the author's knowledge.

3.3. Multi-objective optimization

The goal of this work is to optimize machine learning inference queries. However, like mentioned before, the issue is with balancing both accuracy and execution cost (and possibly other objectives). Multi-objective optimization (MOO) is a field within optimization where multiple objectives functions need to be taken into account. It is a balancing act where optimizing one objective can lead to a direct increase for another objective, as oftentimes objectives conflict.

MOO is a *fuzzy* field, meaning that solutions are not always unequivocally better than others. This is a consequence of the lack of an ordering on $\mathbb{R}^n,\ n > 1$ (one that does not include a prioritization of dimensions at least). An ordering means that for every pair of elements in a set, one of them is larger or equal to ther other. For example, in $\mathbb{N}$, we can say that 7 is larger than 3. In $\mathbb{N}^2$ this is more difficult. Is $\binom{7}{3}$ larger than $\binom{3}{7}$? Unless you specify that the first dimension is more important than the second, you will not be able to define an ordering. If you represent the solution of an MOO problem as a vector of all objective values, then you cannot say that one solution is larger/smaller than another, unless you specify that one objective is more important the other. Given the fact that in optimization we look for the solution of either largest or smallest utility, this is a problem.

This principle has two important consequences:

- More sophisticated ideas of optimality need to be established in order to discuss which solutions are good or bad.
- Possible optimization methods are evaluated on their suitability more by semantic criteria rather than by numerical improvements on a single benchmark.

There are a lot of methods to do MOO, and several survey papers have compiled collections of methods [24][13]. The quality of an MOO method is user-defined and application-dependent however, due to the problems stated above. In the following section we therefore state some important definitions to help us distinguish solutions.

3.3.1. Important definitions for multi-objective optimization

Before we discuss these definitions we first introduce the canonical form of MO optimization problems:

$$\min_{\mathbf{x}} \mathbf{F}(\mathbf{x}) = [F_1(\mathbf{x}), F_2(\mathbf{x}), ..., F_k(\mathbf{x})] \tag{3.4}$$

$$\text{such that } g_j(\mathbf{x}) \leq b_j,\ j = 1, 2, ..., m \tag{3.5}$$

Where $F_k(\mathbf{x})$ represents the kth objective function. A shrewd mathematician will remark that minimizing a vector is tricky (and a poorly-defined) business, and they will be correct: the minimization method will be based upon the chosen MOO method, but that the objectives need to be minimized still needs to be conveyed in the standard form.

Optimality of solution

Different solutions can be found for the same problem. Some solutions compromise on certain objectives in favour of good performance on others, but other solutions might balance all objectives. A ”best” solution therefore does not exist (or is semantically dependent on the problem at hand), but it should go without saying that some solutions are unequivocally worse than others. This leads to the definition of *Pareto Optimality*:

Definition 1 ***Pareto optimality****. A point* $\mathbf{x} \in \mathbf{X}$ *is Pareto optimal iff there does not exist a point* $\mathbf{x}^* \in \mathbf{X}$ *such that* $F_i(\mathbf{x}^*) \leq F_i(\mathbf{x})\ \forall\ 0 \leq i \leq k$ *and* $F_j(\mathbf{x}^*) < F_j(\mathbf{x})$ *for some* j*.*

In more intuitive terms, a solution is Pareto optimal if improving on one objective must lead to decreasing another. The set of Pareto optimal solutions is also known as the *Pareto frontier* or *Pareto boundary*. In some cases it might not be possible to improve on a certain objective at all, irregardless of other objectives. This is also known as *Weak Pareto optimality*:

Definition 2 ***Weak Pareto optimality****. A point* $\mathbf{x} \in \mathbf{X}$ *is weakly Pareto optimal if there does not exist a solution* $\mathbf{x}^*$ *such that* $\mathbf{F}(\mathbf{x}^*) < \mathbf{F}(\mathbf{x})$*.*

In other terms, a point is weakly Pareto optimal if there does not exist a point that improves on every single objective. All points on the Pareto boundary are weakly Pareto optimal, but not necessarily the other way around.

In addition to Pareto optimality, there does exist a concept to denote perfect optimality:

Definition 3 ***Utopia point.*** *A point* $\mathbf{F}^\circ$ *is a utopia point iff for every* $1 \leq i \leq k\ F_i^\circ = \min\{F_i(\mathbf{x}) | \mathbf{x} \in \mathbf{X}\}$

The utopia point is therefore the point where every objective is minimized regardless of the others. Generally, the utopia point in unattainable, as objectives tend to conflict. This concept is in line with the “No free lunch theorem”[11], a paradigm that dictates that optimizing one objective always means decreasing gains on another. In machine learning this for example means that a higher accuracy leads to longer training or computation times, lower robustness, or less explainability. The utopia point does serve use as a comparison point, to give an idea of how far you currently are from perfection.

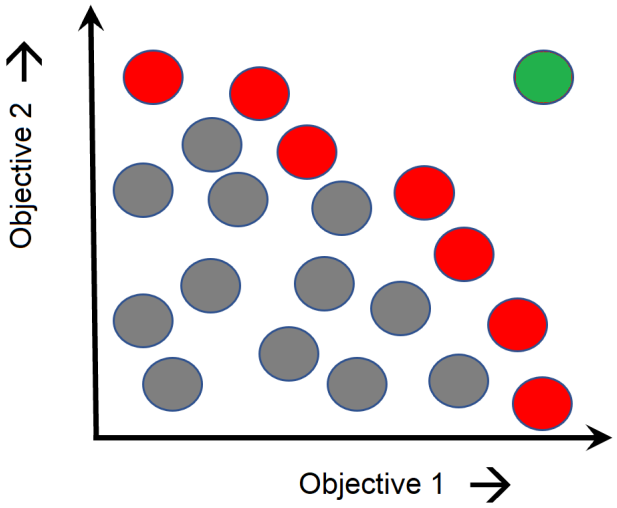


Figure 3.4: A visual representation of the Pareto frontier when trying to maximize objective 1 and 2. The red dots represent the Pareto optimal solutions, forming the Pareto frontier. The grey dots represent non-optimal solutions. The green dot represents the utopia point: it is not a valid solution to this problem, but is the maximally attainable value for both objectives.

Optimality of methods
Next to optimality of solutions, there are also concepts of optimality for MOO methods. Some methods namely guarantee some form of Pareto optimality. This comes in two forms:

Definition 4 ***Optimality of methods.***

- ***(Weak) Pareto sufficiency.*** *A method is (weakly) Pareto sufficient iff it always finds a solution that is (weakly) Pareto optimal.*
- ***Pareto necessity.*** *A method is Pareto necessary iff any Pareto optimal solution can be found using the method.*

These concepts are similar, but generally methods that guarantee Pareto sufficiency are more common than Pareto necessary methods.

Preferences
As mentioned before, MOO is a fuzzy field. Many solutions in a space can be Pareto optimal, and it is upto the user to choose which solution best suits their needs. In order to make a choice, the user has to indicate **preferences**. Preferences are rules that specify some type of literal preference of one objective over another. Typically these preferences are more semantic, and therefore need to be made concrete. This can be done for example by numerical weights, but also by hierarchical ordering. Specifying the right preferences and when in the optimization process is important, but, to quote from the 2004 survey paper by Marler and Arora [24]:

> "Misinterpretation of the theoretical and practical meanings of the weights can make the process of intuitively selecting non-arbitrary weights an inefficient chore."

The paper therefore mentions a few methods for generating weights based on semantic preferences.

Preferences can be articulated at any point in the optimization process. In line with Marler and Arora's paper [24], we divide optimization methods into three categories:

Prior articulation of preferences when the user indicates weights and/or an ordering of importance of objectives before optimization. Methods of this category return a single solution, and a user has to reformulate preferences and repeat the process when a solution is unsatisfactory.

Posterior articulation of preferences where the user chooses a solution from a pre-calculated set, thereby indicating preferences after optimization has occurred. Generally these methods occupy themselves with approximating the Pareto boundary.

No articulation of preferences where no preference is indicated and a single solution is returned.

Which category is best is again problem-dependent. If the optimization problem is easy to solve, it might be preferable to iteratively generate over the Pareto optimal solution and choose afterwards, thereby indicating preferences afterwards. In case of a difficult program, it might be better to indicate preferences beforehand, so that a solver only has to compute a single solution.

4

The optimizer

In the introduction and related work sections Li's work [21] was already mentioned several times. In this paper a non-linear mixed integer program that maximizes accuracy or minimizes cost while bounding the other objective is proposed. Due to the nature of optimization programming, adding new objectives is as simple as adding new equations and amending others. This makes the optimizer highly suitable for extending to multi-objective optimization. In addition to MOO, there are several opportunities in the optimizer that this work attempts to improve on, which will be discussed in section 4.1.3. This chapter is divided into several sections:

Explaining the optimizer: where we explain the two MIPs put forward by Li. We discuss a few examples and places where we can improve on the original.

Amendments to the optimizes: where we discuss the problems identified in the previous section and propose solutions for them.

Towards Multi-objective optimization: before we can actually implement MOO methods, we need to do some preprocessing on the optimizer. Additionally we also introduce a third objective.

Explored MOO methods: a list of the MOO methods we implemented for the optimizer, and a brief explanation how each of them works and processes user input.

The reader is forewarned that the following sections contain a lot of math describing a complicated MIP. Understanding this math is not necessary to understand the value of our contributions, and therefore we allow the reader to read an abridged version of this chapter by only reading the text below.

We can summarize the two optimizers proposed by Li [21] as follows:

- model_opt is an optimizer that computes optimal model selection for an ML inference query using a model zoo. It either optimizes for accuracy while bounding execution cost, or vice versa.
- order_opt is an extension on model_opt that also calculates optimal model order execution on top of optimal model selection.

Both model_opt and order_opt have room for improvement by amending a few equations in the MIP. These amendments attempt to

- Increase transparency of the optimizer
- Discard query plans of poor quality pre-emptively, thereby reducing the size of the search space
- Reformulate equations that likely cause long optimization time

Before we can successfully optimize for multiple objectives, we need to do some preprocessing on the model. These steps include:

- Rescaling the objectives to be in similar range of values (accuracy ranges from 0-1, while execution cost starts at 1 and is unbounded)
- Adding memory footprint as a third objective to that of accuracy and cost

- Proposing a naive way of doing MOO that can generate query plans quickly, but does not generate optimal query plans and does not take user preferences into account

Then we can finally implement the list of MOO methods we consider in the results chapter. These methods are all different in the way they process user input (i.e. preferences), or balance objectives. Some methods guarantee Pareto optimality, and some don't.

4.1. Explaining the optimizer

For the purposes of clarity a short explanation of the optimizer will be discussed, but for more detail the reader is referred to the paper by Li et al. [21]. The paper proposes an optimizer that finds optimal query plans for machine learning inference queries, when there is a model zoo available. To optimize this Li proposes 2 optimizers, both Mixed Integer Programs (MIPs) with most of the contraints linear. The first optimizer, model_opt just provides a selection of models. The second optimizer, order_opt, also finds an ordering that pushes expensive models down the pipeline in addition to model selection. As order_opt is a variation of model_opt, we will first discuss the parts of the optimizers that are identical. Remember the canonical definition of an MIP:

$$\max\ c^T x \tag{4.1}$$

$$\text{such that } Ax \leq b, \tag{4.2}$$

$$x \in \mathbb{N}^n_{\geq 0} \tag{4.3}$$

The idea is that the decision variables x of the program model selection of models for predicates, and the cost function(s) model the accuracy and execution cost of said selection. The matrix A surmises the solution space: the query plans that are able to answer our query q. The optimization software (Gurobi) then searches the solution space for the best query plan according to our objectives (accuracy and cost). We therefore discuss what these equations look like, and how our objectives are defined.

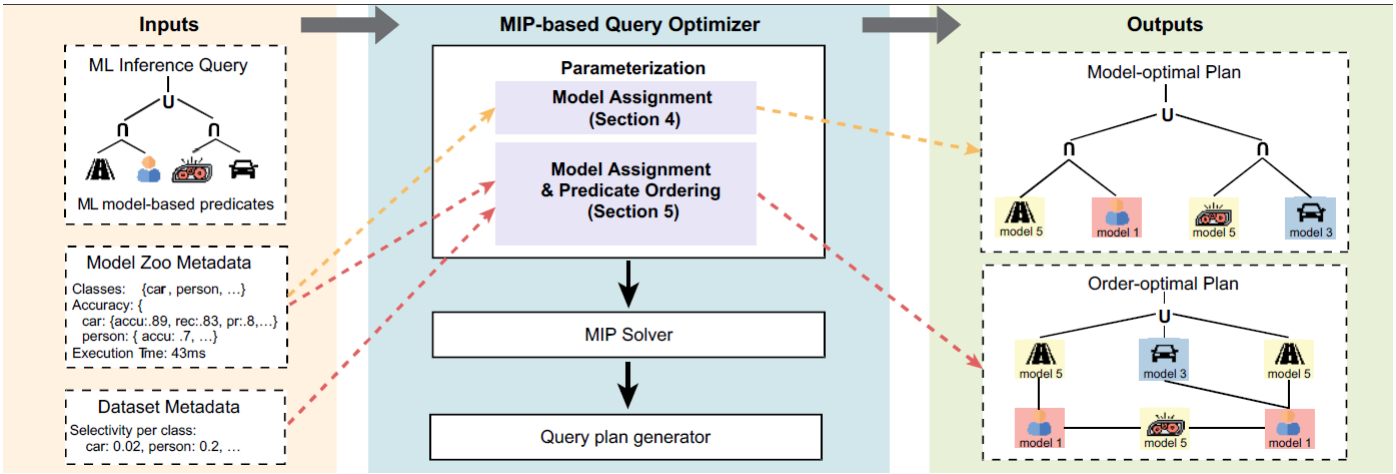


Figure 4.1: Figure 2 from the original paper by Li et al[21] detailing the workflow of the query optimizer. The inputs include a query, a model zoo, and dataset metadata. The optimizer then uses an MIP, where the decision variables can be used to generate a query plan (i.e. model selection and ordering).

Let q be an ML inference query in either CNF or DNF and R a model zoo where the predicates of q are a subset of P. We attempt to find a query plan for q. The decision variables $X_{m,p}$ are defined as

$$X_{m,p} = \begin{cases} 1 \text{ If model } m \text{ is used to answer predicate } p \\ 0 \text{ otherwise} \end{cases} \quad \text{for } m \in M, p \in P \tag{4.4}$$

for both versions. To enforce that every predicate is answered by one model, a restriction needs to be placed on $X_{m,p}$:

$$\sum_{m \in M} X_{m,p} = 1,\ p \in P \tag{4.5}$$

Note how this leaves the possible for one model being selected to answer multiple predicates, but that a predicate is only answered by a single model. These two equations are already sufficient to ensure proper model selection. Now we can define the objective functions, which allow Gurobi to find us the query plan with highest merit.

The execution cost and accuracy are then used as objectives for the model. The combined accuracy calculation remains the same however for both optimizers. The calculation for accuracy is similar to the calculation of probabilities of conjoined or disjoined independent random variables. For conjunctive groups, the accuracy within a predicate group g is calculated as

$$f_{acc}(g) = \prod_{p \in P_g} \sum_{m \in M} A_{m,p} X_{m,p}$$

and for disjunctive groups:

$$f_{acc}(g) = \sum_{p \in P_g} \sum_{m \in M} A_{m,p} X_{m,p} - \prod_{i \in \{p_1, p_2\}} \sum_{m \in M} A_{m,i} X_{m,i} + \prod_{j \in \{p_1, p_2, p_3\}} \sum_{m \in M} A_{m,j} X_{m,j} - \ldots + (-1)^{|P|-1} \prod_{p \in P} \sum_{m \in M} A_{m,p} X_{m,p}$$

The total accuracy is then calculated with the same formula, but using the conjunctive accuracy when the groups are disjunctive, and vice versa. So for DNF queries, where the groups are joined disjunctively, the total accuracy is then calculated as

$$f_{acc}(q) = \sum_{q \in q} f_{acc}(g) - \prod_{i \in \{g_1, g_2\}} f_{acc}(i) + \prod_{j \in \{g_1, g_2, g_3\}} f_{acc}(j) - \ldots + (-1)^{|P|-1} \prod_{g \in q} f_{acc}(g)$$

and for CNF queries, when the groups are conjunctively joined:

$$f_{acc}(q) = \prod_{g \in q} f_{acc}(g)$$

This calculation does assume independence of predicates, and it has not been researched yet in what capacity this influences the true accuracy on data.

4.1.1. Formulation of model_opt

With the equations above we have defined all necessary constraints for model selection. The only thing left is to define our second objective, execution cost f_{cost}. The cost calculation for model_opt is dependent only on if a model is used or not. An 'indicating' variable B_m is therefore needed in order to specify whether a model m is used at all:

$$B_m = \begin{cases} 1 \text{ if } \sum_{p \in P} X_{m,p} \geq 1 \\ 0 \text{ otherwise} \end{cases} \quad \text{for } m \in M \tag{4.6}$$

Note how this constraint is non-linear, while linear constraints are preferred for programming. This issue will be handled in section 4.2. The cost calculation for the total query is then defined as:

$$f_{cost}(q) = \sum_{m \in M} C_m B_m \tag{4.7}$$

Li then adds an objective function by either minimizing cost under a lower bound on accuracy, or maximizes accuracy with an upper bound on cost. In short, model_opt can be summarized as

$$\begin{aligned} &\text{max } f_{acc}(q) \\ &\text{such that } \sum_{m \in M} X_{m,p} = 1, \\ &f_{cost}(q) \leq c \end{aligned}$$

As the accuracy-maximizing-cost-bounding (AMCB) problem, and

$$\begin{aligned} &\min f_{cost}(q) \\ &\text{such that } \sum_{m \in M} X_{m,p} = 1, \\ &f_{acc}(q) \geq a \end{aligned}$$

as the cost-minimizing-accuracy-bounding problem (CMAB) (with a and c user-defined bounds).

Examples
For clarity we give two examples to illustrate the workings of model_opt. For this example we will work with the same dummy model zoo from chapter 1 (table 2.1):

Model name	Model index	Cost	*sentiment*	*person*	*object*
LR	0	5	0.9	0	0
SVM	1	10	0.95	0	0
DNN1	2	20	0	0.92	0.93
DNN2	3	25	0	0.95	0.97
DNN3	4	15	0	0.98	0
DNN4	5	15	0	0	0.99

Table 4.1: A (dummy) model zoo for determining the mentioning of objects and persons in a text, in addition to the sentiment regarding them.

For these examples we do not focus on how to solve the MIP: the intention is to demonstrate how the scores of the query plan are calculated based on a model selection. Therefore we reason about the solution 'greedily' and show how the query plan accuracy and cost are subsequently calculated.

Example 1 *We use a query in DNF form:*

$$q_1 := (person) \vee (sentiment \wedge object) \tag{4.8}$$

We want to find an optimal model selection for this query. For now we optimize for accuracy, and don't set an upper bound on cost. We can reason about the solution without solving the MIP: we just select the models greedily based on their accuracy. We see that for $person$ DNN3 is the best model to select. For $sentiment$ this is SVM, and for $object$ this is DNN4. Therefore we have

$$X_{DNN3,person} = X_{SVM,sentiment} = X_{DNN4,object} = 1 \tag{4.9}$$

And the rest of $X_{m,p} = 0$. Therefore $B_{DNN3} = B_{SVM} = B_{DNN4} = 1$, and $B_{LR} = B_{DNN1} = B_{DNN2} = 0$. This means that

$$f_{cost}(q_1) = 5 \cdot B_{LR} + 10 \cdot B_{SVM} + 20 \cdot B_{DNN1} + 25 \cdot B_{DNN2} + 15 \cdot B_{DNN3} + 15 \cdot B_{DNN4} = 5 + 15 + 15 = 35 \tag{4.10}$$

For the accuracy we first calculate the accuracy of the individual predicate groups $(person)$ and $(sentiment \wedge object)$. For group $g_1 = (person)$ the group accuracy is just $A_{g_1} = A_{DNN3,person} = 0.98$. For the second group $g_2 = (sentiment \wedge object)$ the accuracy is the product $A_{g_2} = A_{SVM,sentiment} \cdot A_{DNN4,object} = 0.95 \cdot 0.99 = 0.9405$. The calculation of the total accuracy of the query is then dependent on the individual group accuracy:

$$f_{acc}(q_1) = A_{g_1} + A_{g_2} - A_{g_1} \cdot A_{g_2} = 0.98 + 0.9405 - 0.98 \cdot 0.9405 = 0.99881 \tag{4.11}$$

Example 2 *For the second example we consider a CNF query:*

$$q_2 = (person) \wedge (sentiment \vee object) \tag{4.12}$$

and optimize over cost. Note how if we do not bound accuracy, the optimizer will just choose the LR model to answer every predicate, thereby achieving 0% accuracy. Therefore we add a lower bound on accuracy:

$$f_{acc}(q_2) \geq 0.9 \tag{4.13}$$

Which would allow the optimizer to choose the cheapest models with non-zero accuracy for every predicate. Therefore

$$X_{LR,sentiment} = X_{DNN1,person} = X_{DNN1,object} = 1 \tag{4.14}$$

and every other $X_{m,p} = 0$. *We calculate the accuracy again by first calculating the accuracy of the individual groups.* $A_{g_1} = A_{DNN1,person} = 0.92$ *and* $A_{g_2} = A_{LR,sentiment} + A_{DNN1,object} - A_{LR,sentiment} \cdot A_{DNN1,object} = 0.9 + 0.93 - 0.9 \cdot 0.93 = 0.993$ *and for the total query accuracy*

$$f_{acc}(q_2) = A_{g_1} \cdot A_{g_2} = 0.92 \cdot 0.993 = 0.91356 \geq 0.9 \tag{4.15}$$

For cost we see that $B_{LR} = B_{DNN1} = 1$ *and* $B_{SVM} = B_{DNN2} = B_{DNN3} = B_{DNN4}$ *= 0 and therefore*

$$f_{cost}(q_2) = C_{LR} + C_{DNN1} = 5 + 20 = 25 \tag{4.16}$$

4.1.2. Formulation of order_opt

As mentioned before, order_opt is an extension upon model_opt that in addition to model selection also calculates model ordering by taking selectivity into account. Selectivity represents a measure of 'rarity' of predicates: the lower the predicate selectivity S_p^P, the rarer the predicate[1]. In this work we assume that $\sum_{p \in P} S_p^P = 1$. In practice the selectivity of predicates can be approximated by calculating the (normalized) occurrence of a class in a dataset. The selectivity of predicates provides the opportunity for taking ordering into account, where the cheaper models are executed early for predicates we expect unlikely to yield positive results. In the introduction we mentioned looking for yellow Volkswagen Polos on the road. Yellow is a rare color for a car (i.e. high selectivity, low S_p^P value), and therefore if we filter images first on the predicate yellow, we might have to only use the "Polo" model on a handful of images. This query plan would therefore likely be quicker than running the two models the other way around, as the true execution time of a model increases with the amount of data that needs to be inferenced on.

To achieve this concept the cost function $f_{cost}(q)$ needs to be reformulated, in order to make the execution cost variable depending on the selectivity of the predicate it is answering. For the ordering a second set of decision variables $O_{p,j}$ is introduced:

$$O_{p,j} = \begin{cases} 1 \text{ If predicate } p \text{ is answered in timestep } j \\ 0 \text{ otherwise} \end{cases} \quad \text{for } p \in P, j \in J \tag{4.17}$$

With J representing the set of timesteps, which is a range from 0 to $|P| - 1$. Similar to $X_{m,p}$, we need to enforce that every predicate is answered within one timestep

$$\sum_{p \in P} O_{p,j} = 1, \ j \in J \tag{4.18}$$

But we also need to also ensure that per timestep only one predicate is answered:

$$\sum_{j \in J} O_{p,j} = 1, \ p \in P \tag{4.19}$$

For computing a cost function using the selectivity of individual predicates, several variables are needed. To start, a binary variable that keeps track whether all predicates in a group have been answered (and therefore the group can be evaluated with certainty):

$$G_{g,j} = \begin{cases} 1 \text{ If all predicates in group } g \text{ have been answered as of timestep } j \\ 0 \text{ otherwise} \end{cases} \quad \text{for } g \in Q, j \in J \tag{4.20}$$

[1] Generally we speak of high selectivity when the S-value is low, as the S-value models the amount of data that matches our predicate. A highly selective predicate means that a small amount of data matches the predicate, and therefore the predicate has a low S-value.

With the set Q representing the groups of query q. $G_{g,j}$ is then constrained as follows:

$$G_{g,j} \geq 1 - |P_g| + \sum_{p \in P_g} \sum_{0 \leq k \leq j-1} O_{p,k},\ g \in q,\ j \in J \tag{4.21}$$

$$G_{g,j} \leq \sum_{0 \leq k \leq j-1} O_{p,k},\ p \in P_g,\ g \in q,\ j \in J \tag{4.22}$$

Additionally, a variable $W_{g,j}$ to quantify the amount of data that has been processed after answering every predicate in a group is needed:

$$W_{g,j} = 1 - G_{g,j} S_g^G,\ g \in q, j \in J \tag{4.23}$$

In this case S_g^G[2] represents the selectivity in groups, which is dependent on the selectivity of predicates within that group. For disjunctive groups, the selectivity is the probability that every predicate returns true:

$$S_g^G = \prod_{p \in P_g} S_p^P,\ g \in q \tag{4.24}$$

And for conjunctive groups, it is the probability that every query predicate returns false:

$$S_g^G = \prod_{g \in P_g} (1 - S_p^P),\ g \in q \tag{4.25}$$

In order to represent selectivity per timestep, we also need to have a measure of selectivity per group per timestep. This is done using continuous variables $H_{g,j}$, which represents the percentage of data being computed at step j when all predicates in group g have been answered. $H_{g,j}$ is defined as

$$H_{g,j} = \prod_{i=0}^{j-1} (1 - \sum_{p \in P_g} O_{p,i}(1 - S_p^P)), \forall g \in q, \forall j \in J \tag{4.26}$$

In the paper by Li [21] an ancillary variable is introduced for $H_{g,j}$ that models the effect when a group has been fully answered. Because this ancillary variable greatly increases computation time we preferred working with slightly less optimal query plans.

The total selectivity per timestep is then computed as

$$S_j^J = \prod_{g \in G} W_{g,j} H_{g,j},\ j \in J \tag{4.27}$$

Before the final cost can be calculated, the cost per timestep and model $R_{m,j}$ needs to be defined:

$$R_{m,j} = S_j^J \sum_{p \in P} X_{m,p} O_{p,j} C_{m,p},\ m \in M,\ j \in J \tag{4.28}$$

And then the final total cost:

$$f_{cost}(q) = \sum_{m \in M} \max_{j \in J} R_{m,j} \tag{4.29}$$

As models should only be executed once if they answer multiple predicates. Similar to model_opt, Li adds a lower bound on accuracy while minimizing cost, or an upper bound on cost while optimizing

[2]The selectivity of a predicate S_p^P is a predefined constant dependent on our dataset, and the selectivity of a group S_g^G is a query-dependent constant, while the selectivity of a timestep S_j^J is variable and dependent on model assignment and ordering. Therefore we use the superscript to indicate which type of selectivity S represents, and the subscript to indicate which member of the set it is.

accuracy. This makes the final program look like:

$$\begin{aligned} &\max f_{acc}(q) \\ &\text{such that} \sum_{m \in M} X_{m,p} = 1, \\ &\sum_{p \in P} O_{p,j} = 1 \\ &\sum_{j \in J} O_{p,j} = 1 \\ &f_{cost}(q) \leq c \end{aligned}$$

For the order_opt version of AMCB and

$$\begin{aligned} &\min f_{cost}(q) \\ &\text{such that} \sum_{m \in M} X_{m,p} = 1, \\ &\sum_{p \in P} O_{p,j} = 1 \\ &\sum_{j \in J} O_{p,j} = 1 \\ &f_{acc}(q) \geq a \end{aligned}$$

for the order_opt version of CMAB.

Examples
Similar for model_opt in section 4.1.1.1, we give an example of the involved calculations for order_opt. Again we use the same dummy model zoo:

Model name	Model index	Cost	*sentiment*	*person*	*object*
LR	0	5	0.9	0	0
SVM	1	10	0.95	0	0
DNN1	2	20	0	0.92	0.93
DNN2	3	25	0	0.95	0.97
DNN3	4	15	0	0.98	0
DNN4	5	15	0	0	0.99

Table 4.2: A (dummy) model zoo for determining the mentioning of objects and persons in a text, in addition to the sentiment regarding them.

From the 'data' that the models have trained on, we derive the following selectivities: $S^P_{sentiment} = 0.4$, $S^P_{person} = 0.5$, $S^P_{object} = 0.1$. We consider the same query as before:

$$q := (person) \wedge (sentiment \vee object) \tag{4.30}$$

and optimize for cost. To once again ensure that our accuracy is somewhat acceptable, we impose a lower bound:

$$f_{acc}(q) \geq 0.9 \tag{4.31}$$

Again we find that for optimal model assignment

$$X_{LR,sentiment} = X_{DNN1,person} = X_{DNN1,object} = 1 \tag{4.32}$$

and for all other $X_{m,p} = 0$. In practice we have to consider only two orderings: first execute LR, and then DNN1, or the other way around. Because the group selectivity is query-dependent and not assignment-dependent we show this calculation first. Because q is in CNF form, the groups are disjunctive. Therefore the group selectivity is the product of the probability of all predicates returning false. Therefore $S^G_{g_1} = S^P_{person} = 0.5$ and $S^G_{g_2} = S^P_{sentiment} \cdot S^P_{object} = 0.4 \cdot 0.1 = 0.04$.

Example 3 *If we execute LR first, we derive the ordering* $O_{sentiment,0} = O_{object,1} = O_{person,2} = 1$ *and the rest of* $O_{p,j} = 0$. *Note how we can choose to first evaluate* $object$ *or* $person$, *but due the lower selectivity* S^P_{object} *and* $object$ *and* $sentiment$ *being in the same group the optimizer will prioritize evaluating* $object$ *first. First we calculate* $G_{g,j}$*:*

$$G_{g_1,0} \geq 1 - |P_{g_1}| + \sum_{p \in P_{g_1}} \sum_{0 \leq k \leq -1} O_{p,k} = 1 - 1 + 0 = 0 \tag{4.33}$$

Which is in line with our expectations that during the first timestep no predicates have yet been evaluated. In fact, as $person$ *does not get evaluated until the last step,* $G_{g_1,0} = G_{g_1,1} = G_{g_1,2} = 0$. *Similar for group 2,* $G_{g_2,0}$. *In the second timestep* $sentiment$ *has been evaluated, but not the entire group* g_2 *is yet evaluated and we see that* $G_{g_2,1} = 0$. *After the first timestep, the entire group has been answered however and therefore we see*

$$G_{g_2,2} \geq 1 - |P_{g_1}| + \sum_{p \in P_{g_2}} \sum_{0 \leq k \leq 1} O_{p,k} = 1 - 1 + O_{sentiment,0} + O_{sentiment,1} + O_{object,0} + O_{object,1} = 1 \tag{4.34}$$

and with the upper bound(s):

$$G_{g_2,2} \leq \sum_{0 \leq k \leq 1} O_{p,k} = O_{sentiment,0} + O_{sentiment,1} = 1 \tag{4.35}$$

$$G_{g_2,2} \leq \sum_{0 \leq k \leq 1} O_{p,k} = O_{object,0} + O_{object,1} = 1 \tag{4.36}$$

Because $G_{g,j} = 0$ *for nearly all group-timestep combinations, we see that* $W_{g,j} = 1$ *for nearly every combination except* $Wg_2, 2$*:*

$$W_{g_2,2} = 1 - G_{g_2,2} \cdot S^G_{g_2} = 1 - 1 \cdot 0.04 = 0.96 \tag{4.37}$$

For $H_{g,j}$ *we need some more involved calculations. Note however the semantic meaning of* $H_{g,j}$, *which is the selectivity of groups per timestep. Because we cannot evaluate* g_1 *until the second timestep, we therefore know* $H_{g_1,0} = H_{g_1,1} = H_{g_1,2} = 1$. *We also know* $H_{g_2,0} = 1$, *but*

$$H_{g_2,1} = H_{g_2,0} \cdot (1 - \sum_{p \in P_{g_2}} O_{p,0}(1 - S^P_p)) = 1 \cdot (1 - (O_{sentiment,0} \cdot (1 - 0.4) - (O_{object,0} \cdot (1 - 0.1) = 0.4 \tag{4.38}$$

And therefore

$$H_{g_2,2} = H_{g_2,1} \cdot (1 - \sum_{p \in P_{g_2}} O_{p,1}(1 - S^P_p)) = 0.4 \cdot (1 - (O_{sentiment,1} \cdot (1 - 0.4) - (O_{object,1} \cdot (1 - 0.1) = 0.04 \tag{4.39}$$

Finally we therefore derive that if

$$S^J_j = \prod_{g=0}^{\#groups} W_{g,j} H_{g,j} \tag{4.40}$$

then $S^J_0 = 1$, $S^J_1 = 0.4$, $S^J_2 = 0.0384$. *Now we can calculate* $R_{m,j}$. *Note how* $R_{m,j} = 0$ *for all models except LR and DNN1.*

$$R_{LR,0} = S^J_0 \cdot \sum_{p \in P} X_{LR,p} O_{p,0} C_{LR,p} = 1 \cdot 1 \cdot 5 + 0 \cdot 0 \cdot \infty + 0 \cdot 0 \cdot \infty = 5 \tag{4.41}$$

And similarly we derive $R_{DNN1,1} = S^J_1 \cdot X_{DNN1,object} \cdot O_{object,1} \cdot C_{DNN1,object} = 0.4 \cdot 1 \cdot 1 \cdot 20 = 5$ *and* $R_{DNN1,2} = 0.768$ *and therefore we finally derive*

$$f_{cost}(q) = \sum_{m \in M} \max_{0 \leq j \leq |P| - 1} R_{m,j} = R_{LR,0} + R_{DNN1,1} = 5 + 5 = 10 \tag{4.42}$$

4.1.3. Considerations

This thesis' main contributions is to extend Li's model with MOO capabilities. However, there are also some other places that the optimizer has room for improvement in terms of computational complexity. As MOO is a more complex optimization problem, it is good practice to therefore attempt to decrease computation time if we see opportunities to do so.

Convexity of the search space

In the background chapter we explained the issues with having product variables in a program. The program proposed by Li contains a lot of product variables in the objective functions (f_{acc} and f_{cost}). There are however several methods for linearizing products of variables, depending on the types of variables [3]. This could leave room for improvements in terms of speeding up the optimization process.

Transparency of the model

The definition of B_m is not linear like mentioned previously in section 4.1.1. In the original version of model_opt this is dealt with by using the big-M method [7]:

$$\begin{cases} \sum_{p \in P} X_{m,p} \leq 1 - \varepsilon + (u - 1 + \varepsilon) \cdot B_m \\ \sum_{p \in P} X_{m,p} \geq B_m + l \cdot (1 - B_m) \end{cases} \tag{4.43}$$

When $\sum_{p \in P} X_{m,p} \in [l, u]$ for every $m \in M$. This version was used in a previous (unpublished) version of model_opt, but amended before first publication as part of our co-authorship.

This method is not ideal. First of all several constants need to be chosen. Second it also removes some of the transparency from the model: an optimization model only solves the model of the problem, not necessarily the problem itself. A model that uses constraints which are easy to understand (and of course model the same concept) should have preference therefore, so possible solutions can be easily checked for appropriateness.

Size of the search space

Next to transparency and non-convexity the size of the search space can also pose problems while optimizing. The size of the search space can be defined as the amount of possible solutions. Many of the solutions are not as ideal as we would like them to be. As mentioned in the example in section 4.1.1.1, it is possible to select a model that has zero accuracy for a predicate. This is somewhat combated by adding a lower bound on accuracy, and in order_opt $C_{m,p}$ is also chosen as infinite for model-predicate combinations of zero accuracy. The lower bound is difficult to tune, especially due to the penalty of a poorly-chosen model for a single predicate becoming smaller as the total amount of predicates in a query increases. Additionally, Implementation-wise, infinity is implemented as a very large number for Gurobi and therefore the solver needs to navigate many solutions it should be discarding. Constraints that would help the solver discard these solutions a priori would be preferable therefore, as this would decrease the size of the search space.

4.2. Amendments to the optimizers

Like mentioned in the previous section, there is room for improvement other than multi-objective optimization. This will therefore be discussed in a separate section.

4.2.1. Amendments to equations

Linearization of B_m

Equation 4.43 can be explained semantically as follows: B_m should be non-zero if and only if at least one $X_{m,p}$ is non-zero for the same m. To ensure that there are two equations needed. The first one ensures that B_m is non-zero when there is at least one $X_{m,p}$ that is non-zero:

$$X_{m,p} \leq B_m, \ \forall m \in M, p \in P \tag{4.44}$$

This ensures that when a predicate is assigned to a model, $X_{m,p}$ 'pushes B_m upwards'. For the purposes of tidiness we also need to enforce that B_m can only be non-zero when there is at least one $X_{m,p}$ that is non-zero. This constraint is arguably unnecessary because of the nature of this being a

minimization problem (and therefore the optimizer would want no unneeded costs to be made), but can be included anyway to decrease the size of the search space:

$$B_m \leq \sum_{p \in P} X_{m,p},\ \forall m \in M \tag{4.45}$$

This equation ensures that in the case of no $X_{m,p}$ being non-zero for a certain m, B_m cannot become non-zero as well. In case there is more than one predicate p for which $X_{m,p}$ is 1, the upper bound is 'dominated' by $B_m \leq 1$, which is implied by it being a binary variable.

The performance of this equation will be explored in the results section, but the point of this amendment is not necessarily to be quicker, but to be more transparent.

Appropriate solutions
Like mentioned before in section 4.1.1.1, the model does not have a constraint that only models which are compatible with a predicate are used (e.g. a model designed for digit recognition should not be used for object detection) and minimizing cost would lead to the overall cheapest model being selected for every predicate (and in turn leading to 0% accuracy). In order_opt this is combated by extending C_m to $C_{m,p}$, where $C_{m,p}$ is C_m when $A_{m,p} > 0$, and infinite elsewhere. While this is an option, it is not ideal. Therefore a constraint enforcing that predicates are only answered by models which have non-zero accuracy on that specific class could decrease the size of the search space. In short, we want the following to hold:

$$A_{m,p} = 0 \Rightarrow X_{m,p} = 0\ \forall m \in M, p \in P \tag{4.46}$$

But since $A_{m,p}$ are given constants, we need to formulate this as a constraint on $X_{m,p}$. Luckily due to $A_{m,p}$ being a given constant, we can use non-linear functions like the ceiling function on it in order to achieve the desired result:

$$X_{m,p} \leq \lceil A_{m,p} \rceil\ \forall m \in M, p \in P \tag{4.47}$$

In this case, when $A_{m,p} = 0$, the ceiling function will make sure that $X_{m,p} = 0$. If $A_{m,p} > 0$, $X_{m,p} \leq 1$ will hold which is already implied by $X_{m,p}$ being a binary variable, and the upper bound will be void.

Linearizing product variables
One critical aspect of (MI)LPs is the usage of only linear equations to describe the solution space. Products of variables are not ideal, and it is not until the release of Gurobi 9 [14] that this particular solver was able to deal with non-linear equations by adding support for non-convex search spaces. The solver documentation digresses however that approximation methods are used, and non-linear equations are significantly more resource-intensive than linear equations. Luckily, using mathematics and properties of the model it is often possible to linearize products when binary variables are involved. This is the case for equations 4.26 and 4.28. The methods for linearizing products of two binary variables or one binary and one continuous variable are explained in more detail in a survey by Ashgari et al [3].

For linearizing 4.26, it should be noted that this equation can be written recursively [3]:

$$H_{g,j} = \prod_{i=0}^{j-1}(1 - \sum_{p \in P_g} O_{p,i}(1 - S_p)) \leftrightarrow H_{g,j} = H_{g,j-1} \cdot (1 - \sum_{p \in P_g} O_{p,j-1}(1 - S_p)), \forall g \in q, 1 \leq j \leq |P| \tag{4.48}$$

Now a variable $Q_{g,p,j}$ can be introduced that takes the value of $H_{g,j}O_{p,j}$:

$$Q_{g,p,j} \leq M \cdot O_{p,j} \qquad \forall g \in q,\ p \in P_g,\ 1 \leq j \leq |P| \tag{4.49}$$

$$Q_{g,p,j} \geq H_{g,j} - (1 - O_{p,j})M \qquad \forall g \in q,\ p \in P_g,\ 1 \leq j \leq |P| \tag{4.50}$$

$$0 \leq Q_{g,p,j} \leq H_{g,j} \qquad \forall g \in q,\ p \in P_g,\ 1 \leq j \leq |P| \tag{4.51}$$

and the following equation can be used henceforth:

$$H_{g,j} = H_{g,j-1} - \sum_{p \in P_g} Q_{g,p,j-1}(1 - S_p) \tag{4.52}$$

[3] This alternative formulation is not mentioned in the original paper by Li, but used in her implementation [20])

Equation 4.28 can also be linearized in a similar fashion as 4.26 by introducing a variable $Y_{m,p,j}$ that takes on the value of $X_{m,p}O_{p,j}S_j$. It is possible to first linearize $X_{m,p}O_{p,j}$ and then use the same procedure as for equation 4.49, but it is also possible to linearize directly:

$$Y_{m,p,j} \leq M \cdot X_{m,p} \qquad \forall m \in M, p \in P, 0 \leq j \leq |P| \tag{4.53}$$
$$Y_{m,p,j} \leq M \cdot O_{p,j} \qquad \forall m \in M, p \in P, 0 \leq j \leq |P| \tag{4.54}$$
$$Y_{m,p,j} \geq S_j - (2 - X_{m,p} - O_{p,j})M \qquad \forall m \in M, p \in P, 0 \leq j \leq |P| \tag{4.55}$$
$$0 \leq Y_{m,p,j} \leq S_j \qquad \forall m \in M, p \in P, 0 \leq j \leq |P| \tag{4.56}$$

This new variable can now be substituted into 4.28:

$$R_{m,j} = \sum_{p \in P} Y_{m,p,j} C_{m,p} \tag{4.57}$$

It should be noted that the calculations for accuracy and total selectivity also include a lot of product variables. In case of the calculation for accuracy, the amount of extra variables that need to be introduced for linearizing so many binary variables is considerable. For the selectivity, linearization of products of continuous variables is needed, which can only be approximated. The gain by linearizing the equations above is already considerable enough to conclude this subsection. The results of adding these equations will be discussed in the final section of this chapter.

4.3. Towards Multi-objective optimization

We have discussed the original optimizers and some improvements that were made upon them. The following two sections will discuss the bulk of this work's contributions, namely the addition of MOO. Some preprocessing has to be done however, and it is fitting to discuss this in its own section.

4.3.1. Amendments to objective functions

For the canonical form for MOO problems (see equation 3.4) all objectives are functions that need to be minimized. This is not ideal for both model_opt and order_opt, which aim to minimize cost but maximize accuracy. This can be easily mitigated however by introducing a new variable aptly named 'accuracy loss':

$$f_{acc\_loss}(q) = 1 - f_{acc}(q) \tag{4.58}$$

It should go without saying that minimizing f_{acc_loss} is equivalent to maximizing f_{acc}. A more difficult issue however is the difference in units that the objectives have. Accuracy indicates a percentage, and might not be uniformly distributed over $[0, 1]$. Cost can range in completely different ways. This means that an optimizer can favour one objective, because optimizing for one will lead to greater improvements over the other. It is therefore appropriate to both scale and normalize objectives. For this the following formula is used, as mentioned in the survey paper by Marler and Arora [24]:

$$F_i^{trans} = \frac{F_i(\mathbf{x}) - F_i^{\circ}}{F_i^{max} - F_i^{\circ}} \tag{4.59}$$

Where $\mathbf{F}^{max}$ refers to the maximum value that the objective takes, and $\mathbf{F}^{\circ}$ the utopia point. This method, also called "true normalization", ensures that all possible values for the objectives are scaled between 0 and 1 in reference to the utopia point and the worst case value while conserving the underlying distribution. For f_{cost} the normalized version then is:

$$f_{cost}^{trans} = \frac{f_{cost} - f_{cost}^{\circ}}{f_{cost}^{max} - f_{cost}^{\circ}} \tag{4.60}$$

But for accuracy we need to be slightly smarter. Note how we can calculate the utopia and max point greedily (see section 4.3.4). These give us values for f_{acc} and f_{cost}. Because we need to normalize f_{acc_loss}, we need to transform equation 4.59 somewhat:

$$f_{acc\_loss}^{trans} = \frac{f_{acc\_loss} - (1 - f_{acc}^{\circ})}{(1 - f_{acc}^{min}) - (1 - f_{acc}^{\circ})} = \frac{f_{acc\_loss} - 1 + f_{acc}^{\circ}}{f_{acc}^{\circ} - f_{acc}^{min}} \tag{4.61}$$

Note how the normalized utopia point is now located at zero. For readability purposes we will now continue under the assumption that optimization is done over normalized objectives, and will not be stated outright everytime.

4.3.2. Addition of a third objective

Having both accuracy and cost as objectives is already sufficient to treat this problem as an MOO problem. One advantage of using MIP for this query optimization problem is that adding new objectives is simple once you find the right equations to add to the program. To demonstrate this we consider a third objective that could be of interest to optimize for, namely that of memory (more specific, storage footprint).

Before we add it to the optimizer, we first need to formalize it as part of the model zoo. Let $D \in \mathbb{N}^{|M|}$ a vector denoting the amount of bytes needed to store model $m \in M$. We add it to the model zoo as $R = (M, I, P, A, C, D)$, and now the dummy model zoo could look something likes this:

Model name	Model index	Cost	Memory	*sentiment*	*person*	*object*
LR	0	5	500	0.9	0	0
SVM	1	10	600	0.95	0	0
DNN1	2	20	1200	0	0.92	0.93
DNN2	3	25	1300	0	0.95	0.97
DNN3	4	15	1000	0	0.98	0
DNN4	5	15	1100	0	0	0.99

Table 4.3: A (dummy) model zoo for determining the mentioning of objects and persons in a text, in addition to the sentiment regarding them.

Note that calculating the total memory footprint of the used models is quite simple. We use the indicator variable B_m and derive:

$$f_{mem}(q) = \sum_{m \in M} B_m D_m \tag{4.62}$$

This objective is the same for both model_opt and order_opt.

4.3.3. Weight calculation

Like mentioned in section 3.3.1.3, many MOO methods use weights in order to indicate user preferences. However, ”choosing numerical weights based on semantic preferences can be a tedious chore” [24]. Therefore there are methods on calculating weights based on semantic preferences. As this work is not about establishing which is the best choice of numerical weights for ML inference queries, we use one of those methods in order to calculate weights for all weighted methods. For this work we are going with a combination of a ranking method and a categorization method [34]. To calculate these weights, order the objectives in categories based on their importance. If c_i^n denotes the number of the category c_i, with the most important categories last, the weights of the objectives in c_i are calculated as follows:

$$w_l = \frac{i}{\sum_j j \cdot \#c_j},\ l \in c_i \tag{4.63}$$

Example 4 *let $a,\ b,\ c,\ d$ objectives ordered as $a \in c_1$, $b, c \in c_2$, and $d \in c_3$. The total sum is then calculated as $1 \cdot 1 + 2 \cdot 2 + 1 \cdot 3 = 8$ and then $w_a = \frac{1}{8}$, $w_b = w_c = \frac{2}{8}$ and $w_d = \frac{3}{8}$*

Unless specified otherwise this method is used to calculate weights for weighted methods.

4.3.4. Greedy solutions

As mentioned before we transform the objectives to be ́truly normalized: Calculating $\mathbf{F}^{max}$ and $\mathbf{F}^{\circ}$ luckily is easy in our case: we can use a greedy strategy to calculate them. In the paper by Li [21] a greedy optimizer is mentioned as comparison to model_opt and order_opt. As the original paper attempted to look at the problem as a version of Knapsack, a more sophisticated greedy method was necessary. In this work we use a simpler version that does not need to heed a bound. The greedy algorithm is used to calculate F° for accuracy as follows:

$$X_{m,p} = \begin{cases} 1 \text{ if } \mathsf{max}(A) = A_{m,p} \\ 0 \text{ elsewhere} \end{cases},\ m \in M,\ p \in P \tag{4.64}$$

and similarly for cost:

$$X_{m,p} = \begin{cases} 1 \text{ if } C_m = \min\{C(n) \text{ where } A_{n,p} > 0\} \\ 0 \text{ elsewhere} \end{cases} \quad , m \in M,\ p \in P \tag{4.65}$$

to make sure that only models are used for their intended purpose. The calculation for memory is the exact same but noted as D rather than C.

The calculation for $\mathbf{F}^{max}$ is then similar. For cost and memory:

$$X_{m,p} = \begin{cases} 1 \text{ if } C_m = \max\{C(n) \text{ where } A_{n,p} > 0\} \\ 0 \text{ elsewhere} \end{cases} \quad , m \in M,\ p \in P \tag{4.66}$$

$$X_{m,p} = \begin{cases} 1 \text{ if } A_{m,p} = \min\{A_{n,p} \text{ where } A_{n,p} > 0\} \\ 0 \text{ elsewhere} \end{cases} \quad , m \in M,\ p \in P \tag{4.67}$$

As calculating all greedy solutions is needed for the MO optimizer they will not be used for baseline comparison.[4]

Greedy-MOO

The greedy method that Li proposes is used as baseline comparison for her work. For our work, we propose a similar greedy method that balances objectives. We use a utility function $U(m,p)$ for a predicate-model combination as

$$U : M \times P \to [0,3]\,,\ U(m,p) = \frac{1 - A_{m,p}}{1 - \min\{A_{n,p} | n \in M\}} + \frac{C_m}{\max C} + \frac{D_m}{\max D} \tag{4.68}$$

The actual model selection is then quite similar to the other greedy methods:

$$X_{m,p} = \begin{cases} 1 \text{ if } U(m,p) = \min\{U(n,p) | n \in M \text{ where } A_{n,p} > 0\} \\ 0 \text{ elsewhere} \end{cases} \quad , m \in M,\ p \in P \tag{4.69}$$

In short, this method selects the model for each predicate that has the highest utility. We do not model greedy model ordering. This method ignores the fact that choosing a model with slightly less utility for a predicate might be more advantageous if it can also be used for answering other predicates. In fact, we can prove that greedy-MOO does not guarantee Pareto optimality by providing a counterexample:

Example 5 *Consider the dummy model zoo with some small amendments:*

Model name	Model index	Cost	Memory	*sentiment*	*person*	*object*
LR	0	5	500	0.9	0	0
SVM	1	10	**900**	0.95	0	0
DNN1	2	20	1200	0	0.92	0.93
DNN2	3	25	1300	0	**0.96**	**0.98**
DNN3	4	15	1000	0	0.98	0
DNN4	5	15	1100	0	0	0.99

We generate the following 'utility' table using the definition of our utility function (equation 4.68). We ignore model-predicate combinations where $A_{m,p} = 0$*.*

Model name	*sentiment*	*person*	*object*
LR	1.58		
SVM	1.59		
DNN1		2.72	2.92
DNN2		2.5	2.29
DNN3		1.25	
DNN4			1.59

[4] Authorial note: I realized very late that this greedy algorithm is incorrect. For normalization purposes the actual MIP is solved for the utopia point, and the greedy solution is used for the worst case solution. There was no time for me to fix this in a better way.

Consider the query $sentiment \vee (person \wedge object)$. The greedy algorithm selects LR to answer $sentiment$, DNN3 to answer $person$ and DNN4 to answer $object$, thereby achieving 0.99702 in accuracy, 35 in cost, and 2600 for memory. However, when we select SVM to answer $sentiment$ and DNN2 to answer both $person$ and $object$, we achieve 0.99704 for accuracy, 35 for cost, and 2200 for memory. We can find a solution that improves on accuracy and memory while keeping the cost the same. In short, the solution found by greedy-MOO is not a Pareto-optimal solution. We therefore conclude that greedy-MOO does not guarantee Pareto optimality.

4.3.5. MOO formalization

Before we move on to the methods that were considered for our optimizer, we end this section by stating the proper MOO formalization of the problem in canonical form. If f_{acc_loss}, f_{cost} and f_{mem} for our objective functions, then MO_model_opt (multi-objective model_opt) can be formulated as follows:

$$\begin{aligned} \text{min } & [f_{acc\_loss}(q), f_{cost}(q), f_{mem}(q)] \\ \text{such that } & \sum_{m \in M} X_{m,p} = 1 \end{aligned}$$

and for MO_order_opt we retrieve a similar formulation:

$$\begin{aligned} \text{min } & [f_{acc\_loss}(q), f_{cost}(q), f_{mem}(q)] \\ \text{such that } & \sum_{m \in M} X_{m,p} = 1, \\ & \sum_{p \in P} O_{p,j} = 1 \\ & \sum_{j \in J} O_{p,j} = 1 \end{aligned}$$

4.4. Explored MOO methods

Now that all definitions, problem statements, background has been established we finally come to one of the main research questions of this work: which MOO method is the best for this model? To answer this question, we consider several methods from the survey paper by Marler and Arora [24] and implement them for our solver. These methods vary in many different ways, which is why we consider so many of them. For more details, further reading, and possible variations the reader is referred to the survey paper [24]. Only methods with prior articulation of preferences are considered due to their ease in implementation and the computational resources available.

In previous chapters we mainly discussed theoretical concepts around MOO. To give some intuition on the difference between methods, we will also provide figures to demonstrate the difference.

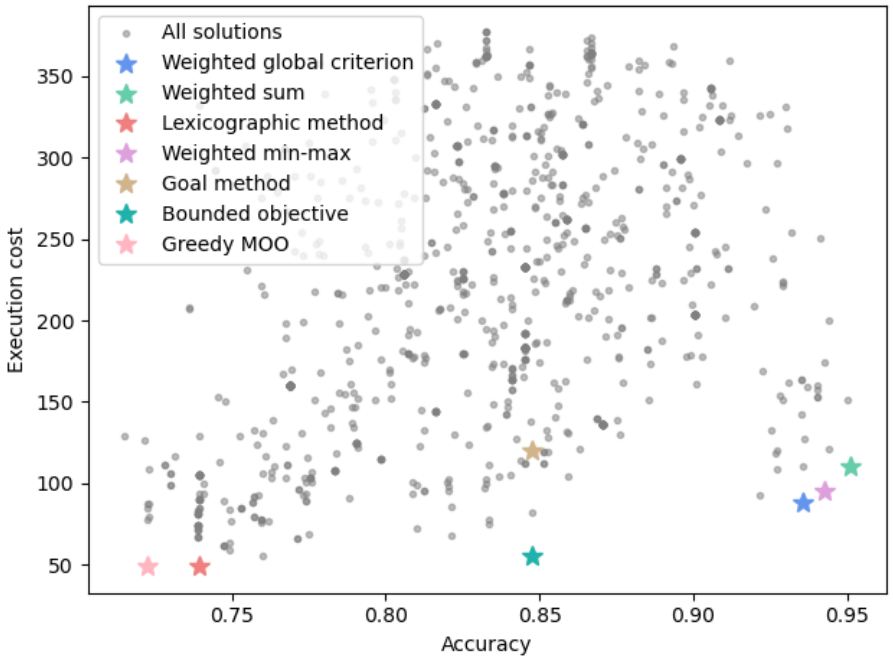


Figure 4.2: The (approximated) solution space of a certain query, plotted as accuracy versus execution cost.

In the above figure we plotted the (approximated) solution space and marked the solutions that were found by several MOO methods. Every dot represents the accuracy and cost of a query plan found for our query. Notice how the methods find different solutions in the solution space. The solution found is dependent on the preference profile we chose (i.e. how the two objectives relate to each other in importance), but methods that use the exact same preference profile still manage to find different solutions. Note especially how some methods, like the goal method, do not find Pareto optimal solutions: we can go more to the right and find a more accurate plan with the same cost, or go down and find a query plan with the same accuracy and lower cost (but in this example we purposefully chose bad goals to demonstrate this). In the following section we will discuss all of these methods and how they work. Additionally we will provide an example of what the MIP looks like for model_opt using this method, and a figure plotting the solutions over several preference profiles. In these figures memory is neglected, due to the legibility of 2D figures.

For the examples we use a hierarchy of accuracy⇒cost⇒memory and weights of $\frac{1}{2}$ for accuracy, $\frac{1}{3}$ for cost, and $\frac{1}{6}$ for memory.

4.4.1. The weighted global criterion

The weighted global criterion is a method that adds all objectives to one single objective function U using weights. U is defined as

$$U = \sum_{i=1}^{k} w_i F_i(\mathbf{x})^p \tag{4.70}$$

Where p is some chosen constant and the weights w_i indicate preference. While the weights indicate preference between objectives, increasing p can also lead to bigger differences between objectives. This method is Pareto sufficient and Pareto necessity has not yet been disproven. The weighted global criterion method uses, like several other methods, a utility function U to indicate the value of a solution.

Example 6 *For the weighted global criterion with $p = 3$ the program for model_opt would look like*

$$\min \frac{1}{2} f_{acc\_loss}(q)^3 + \frac{1}{3} f_{cost}(q)^3 + \frac{1}{6} f_{mem}(q)^3$$
$$\textit{such that} \sum_{m \in M} X_{m,p} = 1$$

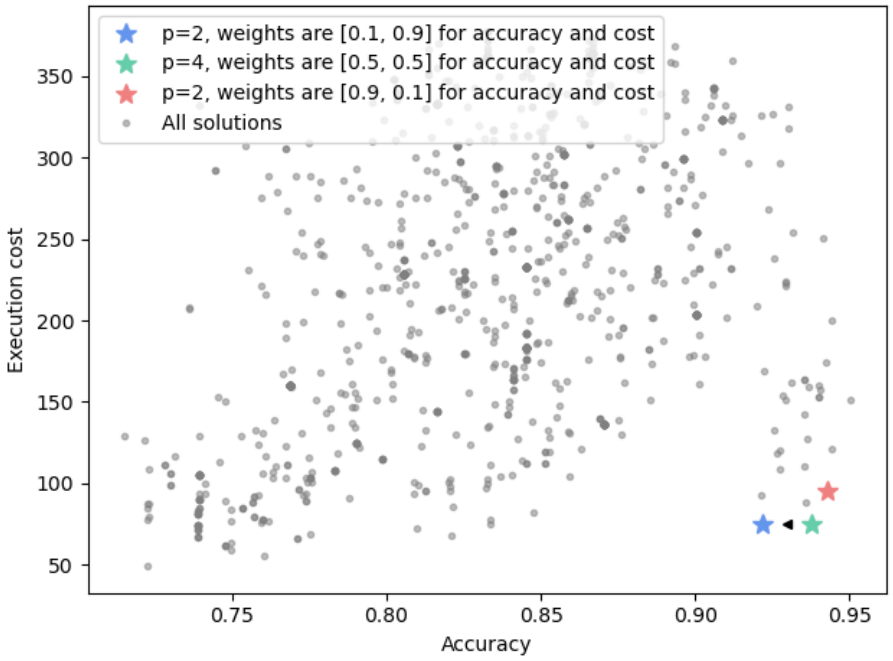


Figure 4.3: The (approximated) solution space of a certain query, plotted as accuracy versus execution cost. Various solutions are plotted using the weighted global criterion, varying over preference profiles. Solutions that fall on the same point are marked with a triangle.

In the above figure we see that the difference between drastically different weights becomes rather small: this is due to the p value, that 'scales' these relative differences. Even with very different weights, the green and blue dot fall on the same solution due to their different p values. This would not work the same way if the objectives were not normalized. Because the objective values now range from 0 to 1 (we optimize over normalized objectives, but our visualizations are over the original values), the p-exponent causes the utility to decrease. If the objectives were larger than one, the exponent would blow the utility up.

In other words, for normalized objectives, a larger p value leads to all possible solutions (by varying over weights) are close together, thereby navigating a small part of the Pareto frontier. For non-normalized objectives that are larger than 1, a large p value leads to the solutions being spread out over the Pareto boundary.

4.4.2. The weighted sum method

The weighted sum is a variation of the weighted global criterion where $p = 1$, thereby making it a linear function. It is therefore a popular method when linear constraints and objectives are desired, and only the weights have to be tuned. Just like the weighted global criterion the method is Pareto sufficient.

Example 7 *For the weighted sum the program for model_opt would look like*

$$\min \frac{1}{2} f_{acc\_loss}(q) + \frac{1}{3} f_{cost}(q) + \frac{1}{6} f_{mem}(q)$$
$$\textit{such that} \sum_{m \in M} X_{m,p} = 1$$

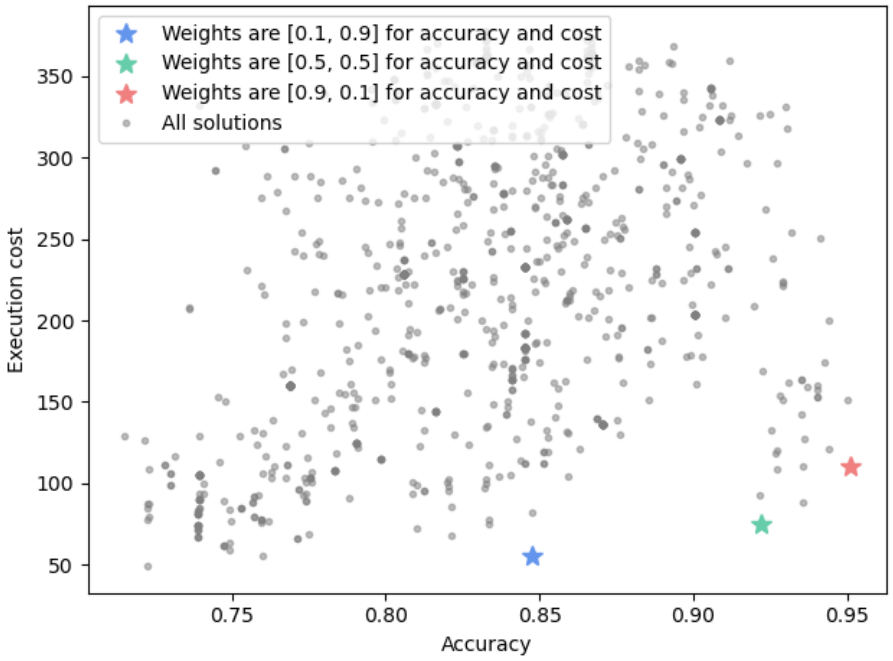


Figure 4.4: The (approximated) solution space of a certain query, plotted as accuracy versus execution cost. Various solutions are plotted using the weighted sum, varying over preference profiles.

In the figure above we see the effect of lowering the p value: the different weights cause the chosen solutions to be spread out over the Pareto boundary, in comparison to the weighted global criterion.

4.4.3. The lexicographic method

The lexicographic method is a method that uses hierarchy for indication of preferences. An ordering of the objectives is given, and every iteration the problem is solved for that objective. After every iteration, the objective value is given to that objective as an upper bound, and the next objective is optimized. This method ensures a solution that is Pareto optimal, but is not suitable when a compromising solution is desired. It also means that the optimizer needs to be run several times, which can lead to longer computation times.

Example 8 *For the lexicographical method the model_opt optimizer needs to be run three times. Using the hierarchy stated above, we first solve*

$$\begin{aligned} &\min\ f_{acc\_loss}(q) \\ &\textit{such that} \sum_{m \in M} X_{m,p} = 1 \end{aligned}$$

if a is the optimal value for accuracy loss (i.e. the objective value) we then solve for cost, adding a bound:

$$\begin{aligned} &\min\ f_{cost}(q) \\ &\textit{such that} \sum_{m \in M} X_{m,p} = 1 \\ &\qquad f_{acc\_loss}(q) \leq a \end{aligned}$$

And if we again obtain c for this program's objective value, we lastly solve

$$\begin{aligned} &\min\ f_{mem}(q) \\ &\textit{such that}\ \sum_{m \in M} X_{m,p} = 1 \\ &\qquad f_{acc\_loss}(q) \leq a \\ &\qquad f_{cost}(q) \leq c \end{aligned}$$

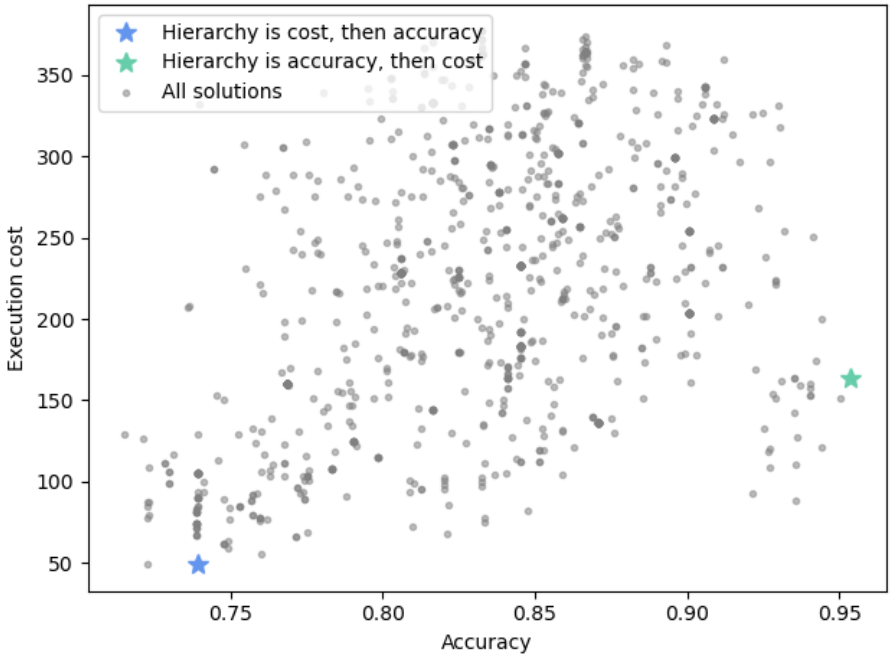


Figure 4.5: The (approximated) solution space of a certain query, plotted as accuracy versus execution cost. Various solutions are plotted using the lexicographic method, varying over preference profiles.

For this problem there are only 2 possible preference profiles: accuracy being the most important objective, or cost. Notice how this results in two solutions that are on near opposite ends on the Pareto boundary. We do therefore see with the blue point that the solution is Pareto optimal: there is a dot to the left with equal cost, but lower accuracy.

4.4.4. The weighted min-max method

In the weighted min-max method (also known as the weighted Tchebycheff method) introduces an ancillary variable λ that serves as an upper bound for every objective:

$$w_i(F_i(\mathbf{x}) - F_i^\circ) \leq \lambda,\ i = 1, ..., k \tag{4.71}$$

after which λ is minimized. The advantage of this method is that is one of the few simpler methods that is necessary for Pareto optimality. The drawback it that it only ensures weak Pareto optimality.

Example 9 *If we use the weighted min-max method for model_opt, we obtain the following program:*

$$\begin{aligned} &\min \lambda \\ &\textit{such that} \sum_{m \in M} X_{m,p} = 1, \\ &\frac{1}{2}(f_{acc\_loss}(q) - f_{acc\_loss}(q)^\circ) \leq \lambda \\ &\frac{1}{3}(f_{cost}(q) - f_{cost}(q)^\circ) \leq \lambda \\ &\frac{1}{6}(f_{mem}(q) - f_{mem}(q)^\circ) \leq \lambda \end{aligned}$$

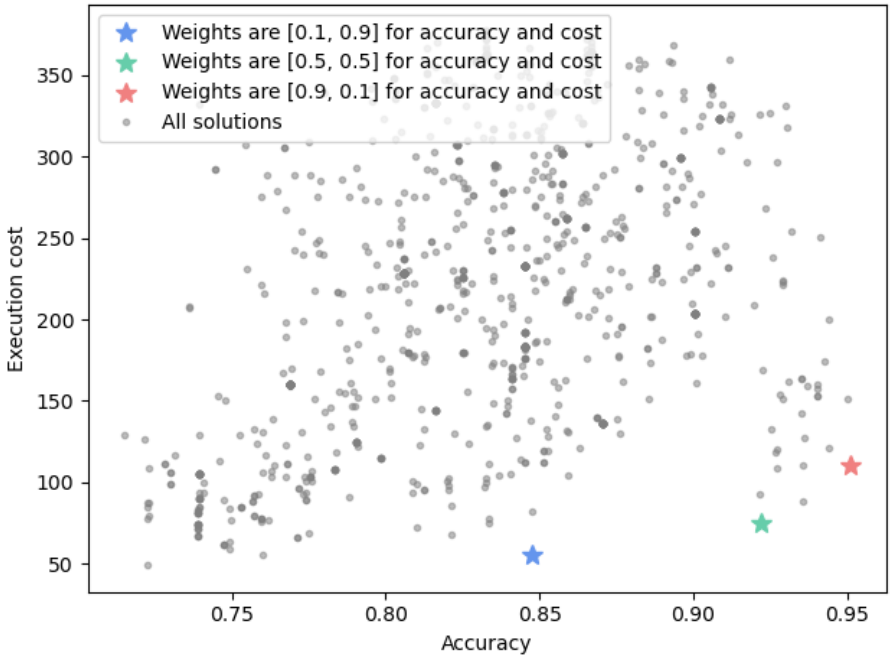


Figure 4.6: The (approximated) solution space of a certain query, plotted as accuracy versus execution cost. Various solutions are plotted using the weighted min-max method, varying over preference profiles.

Notice how we find the exact same solutions as for the weighted sum method (figure 4.4). In this case however, we are not ensured that for other queries only Pareto optimal solutions are found.

4.4.5. Exponential weighted criterion

The exponential weighted criterion is another method in which all objectives are combined into a utility function U:

$$U = \sum_{i=1}^{k} (e^{pw_i} - 1)e^{pF_i(\mathbf{x})} \tag{4.72}$$

This method is both sufficient and necessary for Pareto optimality, but not ideal due to the risk of numerical overflow in the exponent.

Example 10 *For the exponential weighted criterion the program if $p = 3$ for model_opt would look like*

$$\begin{aligned} &\min (e^{\frac{3}{2}} - 1)e^{3 \cdot f_{acc\_loss}(q)} + (e^{\frac{3}{3}} - 1)e^{3 \cdot f_{cost}(q)} + (e^{\frac{3}{6}} - 1)e^{3 \cdot f_{mem}(q)} \\ &\textit{such that} \sum_{m \in M} X_{m,p} = 1 \end{aligned}$$

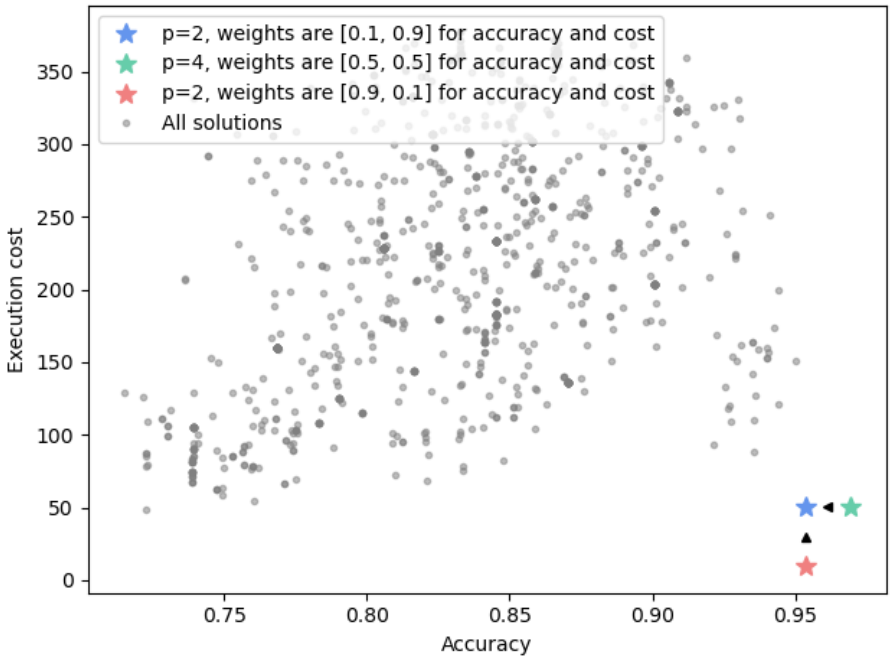


Figure 4.7: The (approximated) solution space of a certain query, plotted as accuracy versus execution cost. Various solutions are plotted using the exponential weighted criterion, varying over preference profiles.

Notice how the exponential weighted criterion finds only the same solution three times, over widely varying preference profile. We do digress that for this figure we had to enable a certain approximation method (infeasibility relaxation) that reduces the quality of solutions, meaning that the found solutions might actually be infeasible but feasible in the approximation. For the actual experiments this approximation method was not used, but for the purpose of visualization it was used for this example.

4.4.6. Weighted product method

Similar to the previous, the weighted product method also combines all objectives into a single function U:

$$U = \prod_{i=1}^{k} F_i(\mathbf{x})^{w_i} \tag{4.73}$$

Similar to the exponential weighted criterion, this method can be computationally expensive.

Example 11 *We formulate the program for model_opt using the weighted product method. For this method we have to translate the weights to integers, for which we have taken the relative inverse weight ($w_i' = \frac{w_i}{min(w)}$) and so:*

$$\text{min } f_{acc\_loss}(q)^3 \cdot f_{cost}(q)^2 \cdot f_{mem}(q)^1$$
$$\textit{such that } \sum_{m \in M} X_{m,p} = 1$$

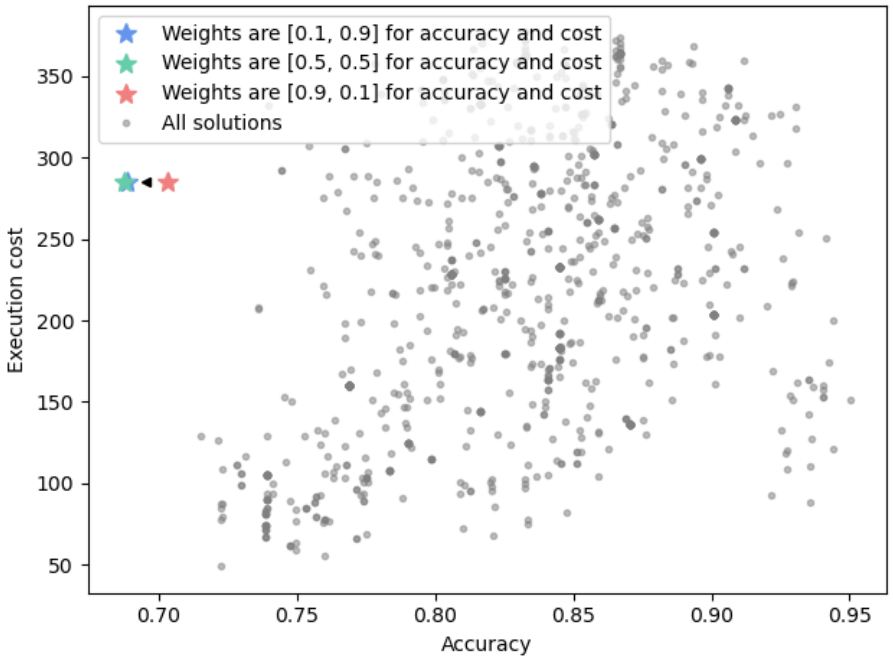


Figure 4.8: The (approximated) solution space of a certain query, plotted as accuracy versus execution cost. Various solutions are plotted using the weighted product, varying over preference profiles.

For the weighted product method we encountered the same problems as for the exponential weighted criterion.

4.4.7. Goal programming method

Goal programming methods minimize the distance of the solution to some predefined goal **b**. This can be done in multiple ways, but the method used in this work is the following:

$$\min \sum_{i=1}^{k} (d_i^+ + d_i^-) \tag{4.74}$$

$$\text{such that } F_j(\mathbf{x}) + d_j^+ - d_j^- = b_j,\ j = 1, ..., k, \tag{4.75}$$

$$d_j^+, d_j^- \geq 0,\ j = 1, ..., k, \tag{4.76}$$

$$d_j^+ \cdot d_j^- = 0,\ j = 1, ..., k \tag{4.77}$$

This method does not guarantee any kind of Pareto optimality, but can have great potential based on a well-chosen goal. An obvious choice for example would be to choose the utopia point as the goal, as long as it can be calculated easily. Setting a Pareto optimal point or the Utopia point as the goal also guarantees Pareto optimality[25]. It should be noted that setting the Utopia point leads to a method that is functionally identical to the weighted sum method (with equal weights).

Example 12 *If we take goals of 0 for accuracy loss, 0.1 for cost, and 0.2 for memory (these goals are*

normalized), we obtain the following program for model_opt using the goal method:

$$\begin{aligned}
&\min \sum_{i=1}^{3}(d_i^+ + d_i^-) \\
&\textit{such that } \sum_{m\in M} X_{m,p} = 1 \\
&\quad f_{acc_loss}(q) + d_1^+ - d_1^- = 0 \\
&\quad f_{cost}(q) + d_2^+ - d_2^- = 0.1 \\
&\quad f_{mem}(q) + d_3^+ - d_3^- = 0.2 \\
&\quad d_j^+, d_j^- \geq 0,\ j = 1, ..., 3 \\
&\quad d_j^+ \cdot d_j^- = 0,\ j = 1, ..., 3
\end{aligned}$$

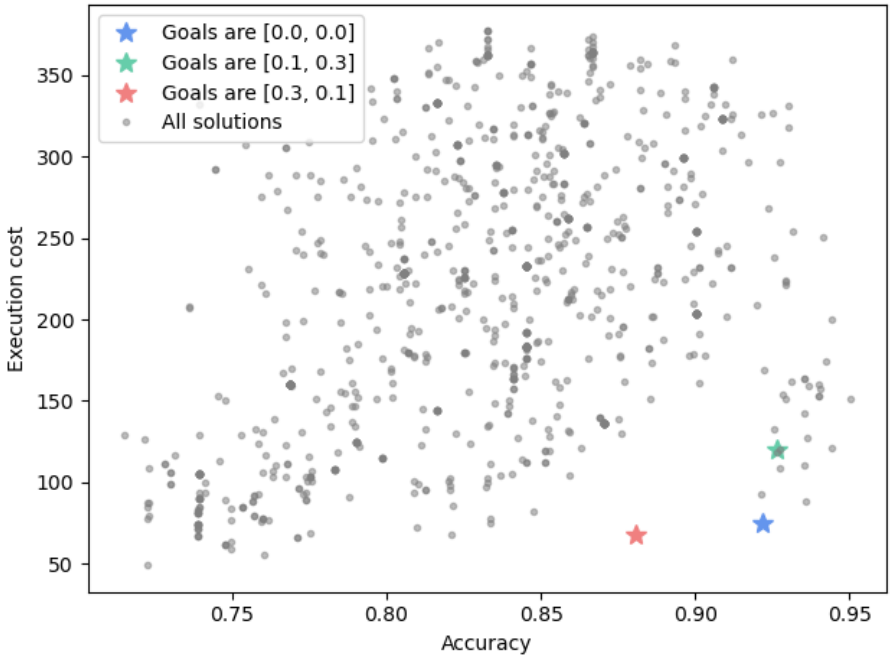


Figure 4.9: The (approximated) solution space of a certain query, plotted as accuracy versus execution cost. Various solutions are plotted using the goal method, varying over preference profiles.

Notice how a good choice of a goal leads to a Pareto optimal solution, in case of the blue and red dot. The green dot is however not Pareto-optimal.
The goal method is popular due to its usability and therefore we will also be considering several variations on it.

The first variation on the goal method we are considering is the Archimedean goal programming method [5], which is also known as the weighted goal programming method [25]:

$$\min \sum_{i=1}^{k} w_i(d_i^+ + d_i^-) \tag{4.78}$$

$$\text{such that } F_j(\mathbf{x}) + d_j^+ - d_j^- = b_j,\ j = 1, ..., k, \tag{4.79}$$

$$d_j^+, d_j^- \geq 0,\ j = 1, ..., k, \tag{4.80}$$

$$d_j^+ \cdot d_j^- = 0,\ j = 1, ..., k \tag{4.81}$$

This allows the user to indicate preferences both by goals and weights. When the utopia point is set, this method is functionally identical to the weighted sum method. Just like the regular goal programming method, the method guarantees Pareto optimality when a Pareto optimal goal is set.

Example 13 *By taking the same goals, we obtain the following program for model_opt using the Archimedean goal method:*

$$\begin{aligned}
&\min \frac{1}{2}(d_1^+ + d_1^-) + \frac{1}{3}(d_2^+ + d_2^-) + \frac{1}{6}(d_3^+ + d_3^-) \\
&\textit{such that } \sum_{m \in M} X_{m,p} = 1 \\
&f_{acc_loss}(q) + d_1^+ - d_1^- = 0 \\
&f_{cost}(q) + d_2^+ - d_2^- = 0.1 \\
&f_{mem}(q) + d_3^+ - d_3^- = 0.2 \\
&d_j^+, d_j^- \geq 0, \ j = 1, ..., 3 \\
&d_j^+ \cdot d_j^- = 0, \ j = 1, ..., 3
\end{aligned}$$

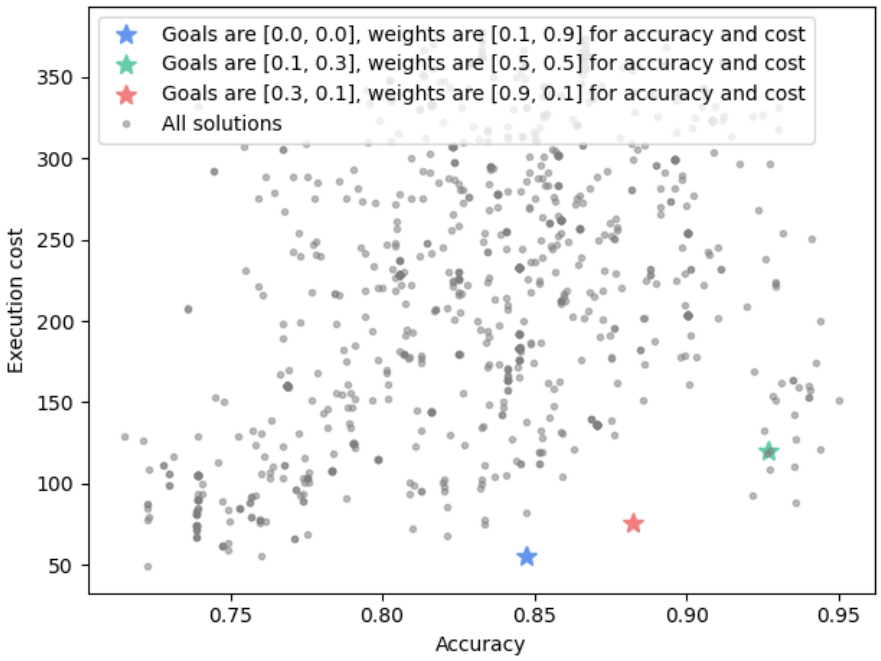


Figure 4.10: The (approximated) solution space of a certain query, plotted as accuracy versus execution cost. Various solutions are plotted using the Archimedean goal method, varying over preference profiles.

Notice how by varying over the weights we can actually find very different points than the regular goal method.

The other variation we consider is a combination of the weighted min-max method and the goal method, also known as the goal attainment method [9]:

$$\min_{\mathbf{x} \in \mathbf{X}, \lambda} \lambda \tag{4.82}$$

$$\text{such that } F_i(\mathbf{x}) - w_i \lambda \leq b_i \qquad i = 1, 2, ..., k \tag{4.83}$$

When the utopia point is set as the goal this method is functionally identical to the weighted min-max method, and therefore does not guarantee Pareto optimality when setting a clever goal, unlike the other discussed goal programming methods.

Example 14 *Using the same goals we obtain the following program for model_opt using goal attainment method:*

$$\begin{aligned} &\min \lambda \\ &\textit{such that} \sum_{m \in M} X_{m,p} = 1 \\ &f_{acc\_loss} - \frac{1}{2}\lambda \leq 0 \\ &f_{cost} - \frac{1}{3}\lambda \leq 0.1 \\ &f_{mem} - \frac{1}{6}\lambda \leq 0.2 \end{aligned}$$

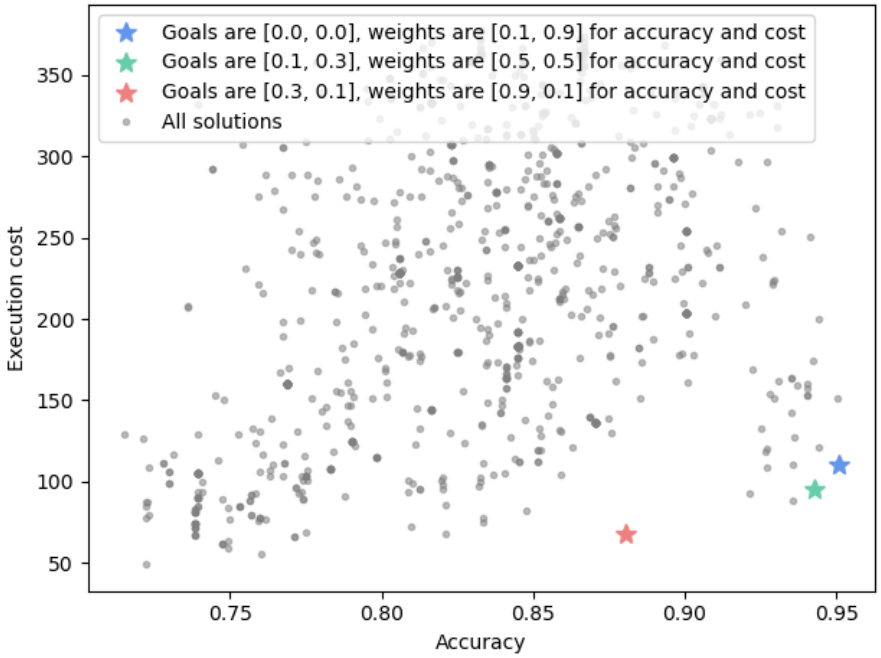


Figure 4.11: The (approximated) solution space of a certain query, plotted as accuracy versus execution cost. Various solutions are plotted using the goal attainment method, varying over preference profiles.

Again we see that by using a variation of the goal method we can find different solutions.

4.4.8. Bounded objective method

The bounded objective method is the method used in the paper by Li [21]. In this method, one single objective is chosen as the objective function, with all other objective functions receiving upper and lower bounds. It is easy to use, but like the goal method, it does not guarantee any kind of optimality and is even more vulnerable to poorly chosen parameters. For example, a well chosen goal can lead to Pareto optimality, but poorly chosen bounds can lead to infeasibility or no kind of Pareto optimality.

Example 15 *Because we set accuracy as the most important objective, we optimize this objective and impose bounds on the other objectives. If we set 0 as the lower bound of both objectives, and 0.1 as the upper bound for cost and 0.2 for memory (bounds are normalized) we obtain the following program*

for model_opt:

$$\min f_{acc\_loss}(q)$$
$$\textit{such that} \sum_{m \in M} X_{m,p} = 1$$
$$0 \leq f_{cost} \leq 0.1$$
$$0 \leq f_{mem} \leq 0.2$$

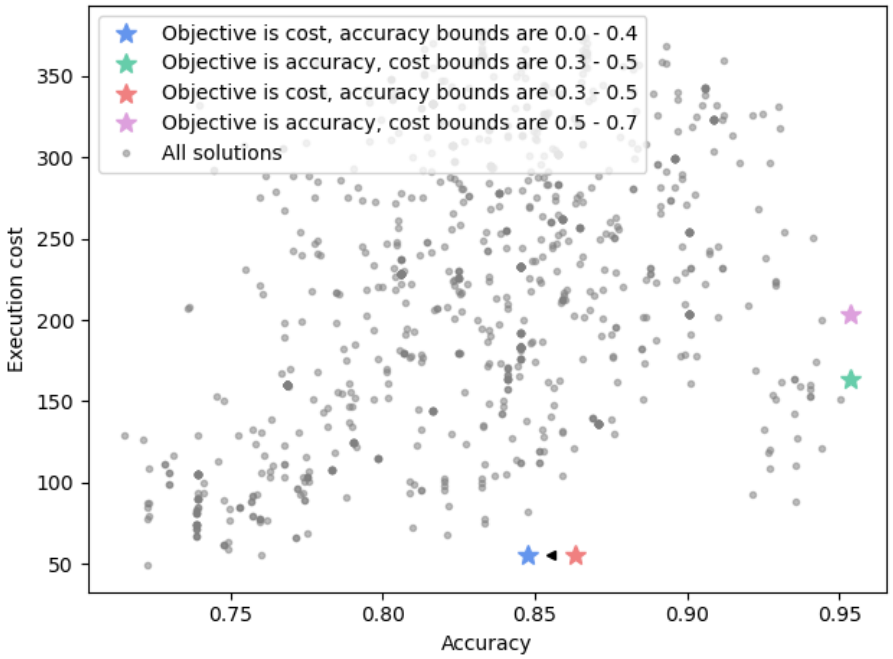


Figure 4.12: The (approximated) solution space of a certain query, plotted as accuracy versus execution cost. Various solutions are plotted using the bounded objective method, varying over preference profiles.

In this figure we have purposefully set bad bounds to demonstrate the difficulty with this method. In case of the pink versus the green solution, they have equal accuracy. Our optimizer would not to distinguish them however when optimizing for accuracy, as the optimizer only cares for optimizing one objective (in this case accuracy). As long as the cost of both solutions falls within the bound set on cost, the optimizer would not choose green over pink, even though green is an objectively better query plan.

4.4.9. Summary

In the previous section we have introduced many MOO-methods. Because these methods both differ and have similarities, we arrange them in some structured tables. First we can group methods by their way of dealing with preferences:

	Preference method				
Method	Weights	p-exponent	Hierarchy	Goal	Bounds
Weighted global criterion	x	x			
Weighted sum	x				
Lexicographic			x		
Weighted min-max	x				
Exponential weighted criterion	x	x			
Weighted product	x				
Standard goal programming				x	
Archimedean goal programming	x			x	
Goal attainment programming	x			x	
Bounded objective					x
Greedy MOO					

From the examples in the previous section we can already conclude that some ways of indicating preferences are not ideal. Because we are working normalized objectives, the p-exponent tends to shrink differences among objectives, rather than enlarge them. Using a hierarchy is also not ideal, as there are only 2 options for indicating preferences when using 2 objectives (6 when using 3 objectives), and these preferences do not allow for good compromise solutions (solutions that try to balance objectives, rather than greatly favour one). A goal is a good way of indicating preferences, but some knowledge of the solution space is needed in order to achieve good solutions (or just setting the utopia point as the goal, with which you are usually emulating other methods). Bounds are in general a good way of indicating preferences for our problem, but the bounded objective method does not guarantee Pareto optimality which is far from ideal.

Next we consider some other characteristics typically encountered in the aforementioned MOO-methods:

Method	**Linear method**	**Utility function**	**Utopia point**	**Pareto optimality**
Weighted global criterion	Possible	x	No	Yes
Weighted sum	x	x	No	Yes
Lexicographic	x		No	Yes
Weighted min-max	x		Yes	Weak
Exponential weighted criterion		x	No	Yes
Weighted product		x	No	Yes
Standard goal programming	x		Possible	Goal-dependent
Archimedean goal programming	x		Possible	Goal-dependent
Goal attainment programming	x		Possible	Weak, goal-dependent
Bounded objective	x		No	No
Greedy MOO	x	x	No	No

In general we would prefer a linear method, as both model_opt and order_opt can get quite computationally expensive when considering queries of a large amount of predicates. Utility functions are a great way of processing preferences, But there aren't many methods that are also linear. Some methods make use of the utopia point, or give the option to (such as the goal methods). As the utopia point is the closest we have to perfect optimality, it can serve as a good reference point. Most important is the Pareto optimality column. Luckily most methods offer some kind of Pareto optimality, with only the bounded objective method and greedy MOO being unable to do so.

5

Results

In the following chapter we demonstrate the performance of the amendments to model_opt and order_opt in section 4.2, the performance of the MO optimizer proposed in section 4.3.5 using the methods from section 4.4. Finally we also investigate the performance of the MO optimizer on real-life datasets using real models.

The questions we would like to answer by conducting these experiments are therefore:

1. Do the amendments proposed in section 4.2 affect the optimizers' runtime, and are all amendments beneficial?
2. Do these amendments improve on the actual computational complexity of the optimizer?
3. Which MOO method is the best choice for model_opt and order_opt?
4. How do the MO-optimizer-generated query plans perform on real-life data using real models in comparison to naive methods?

Because the first three questions can be answered without actually running the query plans on real data, we made the choice to run these experiments on a synthetic model zoo. This model zoo can be found on Li's github page [20] and is based on the COCO [22] dataset. The models in the accuracy and cost tables do not actually exist, and therefore the inferencing capabilities can be quite unrealistically varied (i.e. models can classify a small number of widely unrelated classes, with greatly varying accuracy). This makes it a more interesting space for the optimizer to navigate. The selectivity however is based on the actual dataset.

5.1. Equation amendments

First we consider the performance of the equation amendments proposed in section 4.2. To do this we perform a thorough ablation study to investigate the impact of the amendment on the runtime of the optimizer.

For model_opt we tested the following ablations:

1. No amendments
2. New linearization of B_m (equation 4.44 and 4.45)
3. Upper bound on accuracy (equation 4.47)
4. Upper bound on accuracy and new linearization of B_m

and for order_opt we tested:

1. No amendments
2. Linearization of $R_{m,j}$ (Equation 4.57)
3. Upper bound on accuracy (Equation 4.47)
4. Linearization of $H_{g,j}$ (Equation 4.52)

5. Upper bound on accuracy and linearization of $H_{g,j}$
6. Linearization of $H_{g,j}$ and linearization of $R_{m,j}$
7. Upper bound on accuracy and linearization of $R_{m,j}$
8. Upper bound on accuracy and linearization of $H_{g,j}$ and linearization of $R_{m,j}$

To test this we randomly generated 10 queries of 8 predicates. Because we are trying to see the difference between ablations, it is not necessary to run the optimizer on a large amount of queries. The full list of queries can be found in the appendix. Every query was optimized for respectively accuracy and cost over every single ablation for model_opt and order_opt.

Li's optimizer needs a lower bound on accuracy or upper bound on cost when optimizing, as mentioned before (otherwise the results are not very interesting). This bound is chosen halfway between the optimal and worst value of that objective. This bound does influence the computation time, but testing several bounds was out of scope for this work. For the impact of bound setting the reader is referred to Li's original paper [21].

To get a good estimate of the actual runtime, we ran every experiment 10 times. Computation time was measured by Gurobi's own runtime attribute, which gave a more consistent measurement than using Python's time modules. In addition to runtime 'work' was also considered, which is one of Gurobi's internal measurement of optimization effort, and is consistent between runs that have successfully finished optimization (work is however not a perfect predictor of optimization time). From the Gurobi documentation[1], one work unit corresponds very roughly to one second on a single thread. As optimization time can be very long for some queries, we marked a 30 minute cutoff as 'dnf' (did not finish). This is certainly not preferable, but limited hardware was available for the experiments on the optimizer and concessions had to be made.

Next to computational performance, we also looked at a 'bug report'. For some queries the optimizer could not find a solution, even though the greedy solution was available. This appeared to be an issue with the optimizer software, as parameter tuning or setting a starting solution mitigated the issue, but did not prevent it in every case. Therefore the Gurobi optimizer output status is also considered, which returns a 2 when optimization has occurred successfully, a 3 if the model is deemed infeasible (which is our 'bug', as we know that the greedy solution must exist), and a 9 if the optimization timeout was reached. Luckily output code 3 is rare, and was not encountered in most of our experiments. For more output codes the reader is referred to the Gurobi documentation[2]. In case of an infeasibility error, the greedy approach for accuracy was used as a starting solution, but the solver did not always manage to move to a different solution, and even the starting solution did not guarantee that the infeasibility would be resolved. In case of a timeout, the optimizer chooses the current solution of the solver and runtime is truncated to 30 minutes.

The hardware used for the experiments in this section is a Windows environment with 16 GB RAM, an Intel 8th generation i7 processor with 4 cores and no GPU. The TU Delft cluster was available to the author but Gurobi could not be installed on it, therefore they had to resort to personal hardware for query plan generation.

To compare the results, we are viewing them averaged over all queries. While it is more enlightening to look at individual queries, the figures are hard to read and demonstrate our conclusions in a less coherent way. The figures without averaging can be found in the appendix.

[1] https://www.gurobi.com/documentation/10.0/refman/work.html
[2] https://www.gurobi.com/documentation/9.5/refman/optimization_status_codes.html

(a) Computation time in ms

(b) Work

Figure 5.1: Results for model_opt for several queries over all ablations, optimized over both objectives

We see that the barplots have a fairly small standard deviation (indicated by the yellow error bar). Like the axes, the error bars also scale logarithmically, so for the leftmost bars there is a larger standard deviation than appears in comparison to the rightmost ablations.

Overall we see that when optimizing over cost, both the linearization of B_m and the upper bound on accuracy offer improvement, having nearly a third of the original computation time. When optimizing over accuracy, we see that mainly the upper bound on accuracy offers great improvements. The time-out results have been kept in however (explaining why the average for some of the ablations is so large), meaning that is hard to estimate how large the actual improvement is. All these results are also corroborated by figure 5.1b, which shows the 'work'.

For good measure we check the error report:

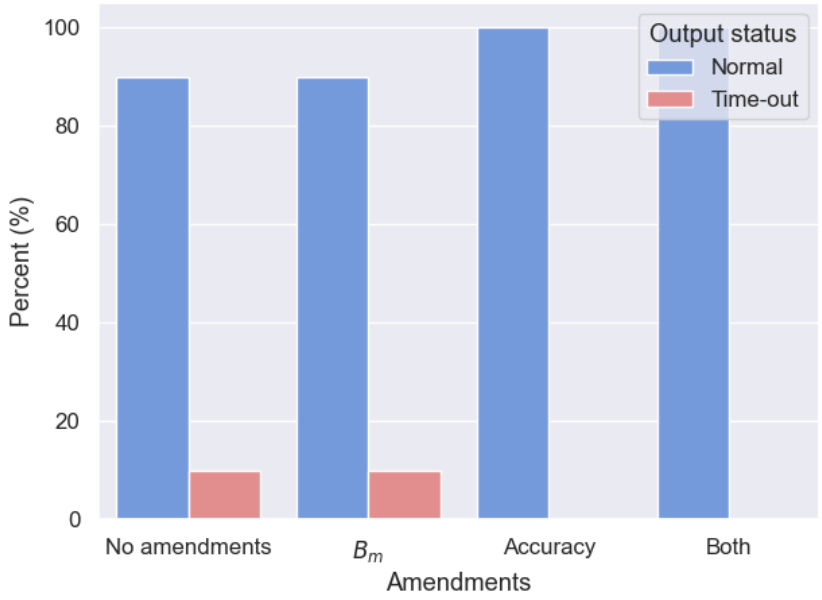


Figure 5.2: Bug report for the different ablations over model_opt

We see that the ablations without amendments and the one with only the linearization of B_m have a few timeouts, about 10% of the results. Again the bug report reaffirms that the upper bound on accuracy has great improvement on runtime. As the main goal of the alternative linearization of B_m was transparency over computational speed, we conclude that both amendments benefit model_opt.

Next we look at the results for order_opt. From the paper by Li [21] we already know that order_opt is significantly more computationally expensive than model_opt. Many of the results had timeouts, and it is difficult to meaningfully represent the results therefore. Removing the timeouts results in skewed graphs. therefore we decided to keep these results in, with the runtime of truncated results kept at the cut-off of 30 minutes. The graphs therefore might not entirely accurately represent the true optimization times, but the merit of the amendments is still evident.

First we consider the bug report:

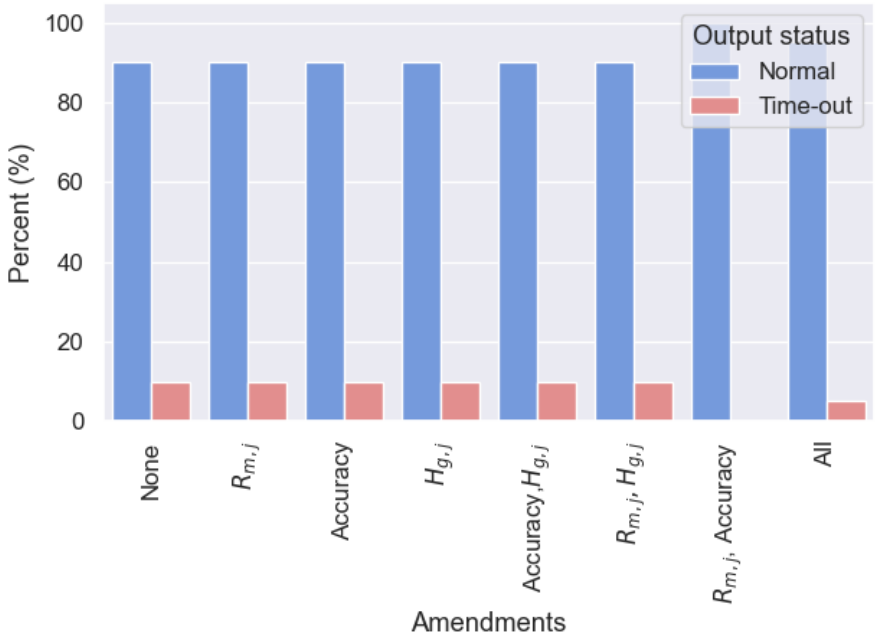
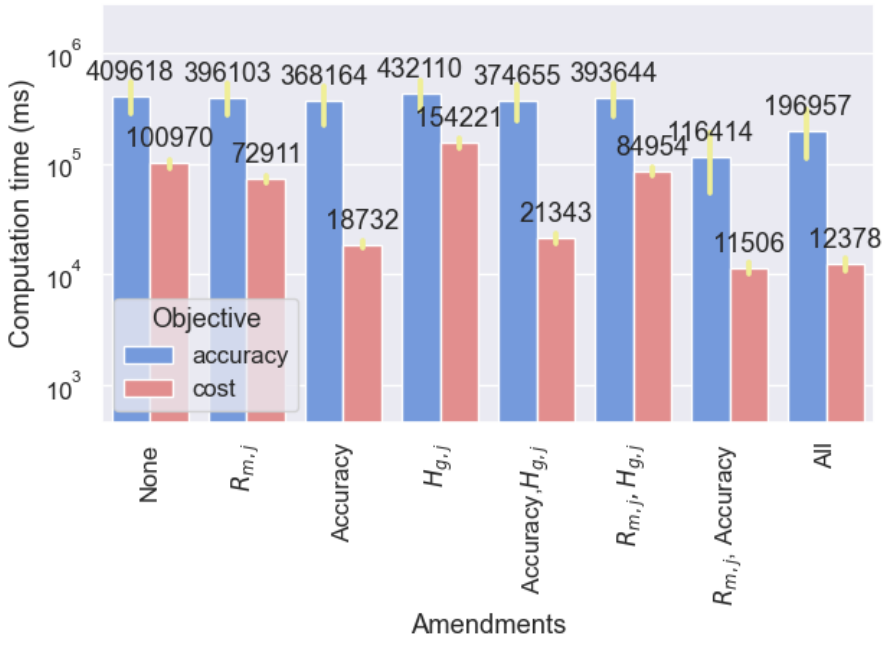
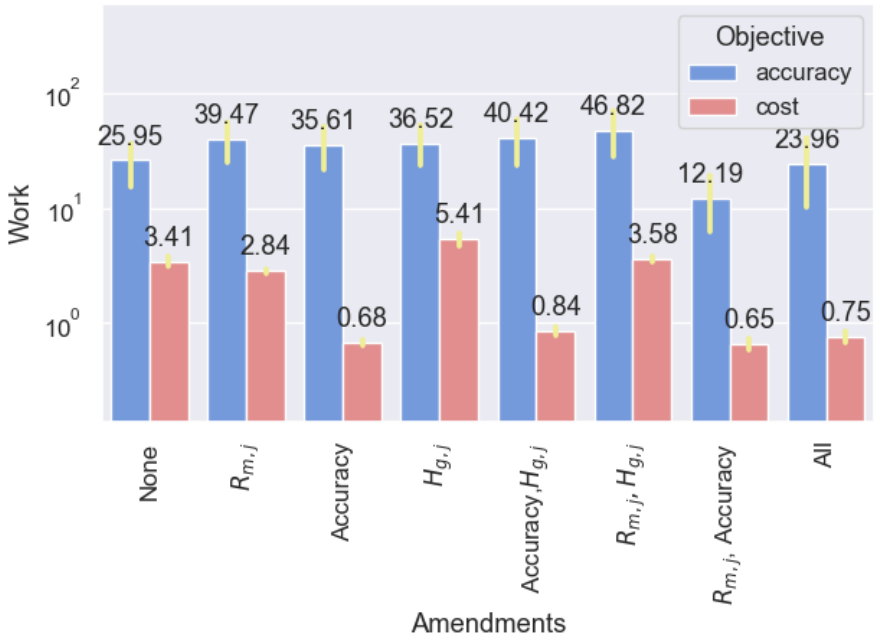


Figure 5.3: Bug report for the different ablations over order_opt

We see a large number of timeouts for every ablation, but there are ablations with fewer timeouts. Interesting is that we see that the first-to-last ablation has no timeouts, which includes all amendments except the linearization of $H_{g,j}$. Next we compare the runtime and work:



(a) Computation time in ms



(b) Work

Figure 5.4: Results for order_opt for several queries over all ablations, optimized over both objectives

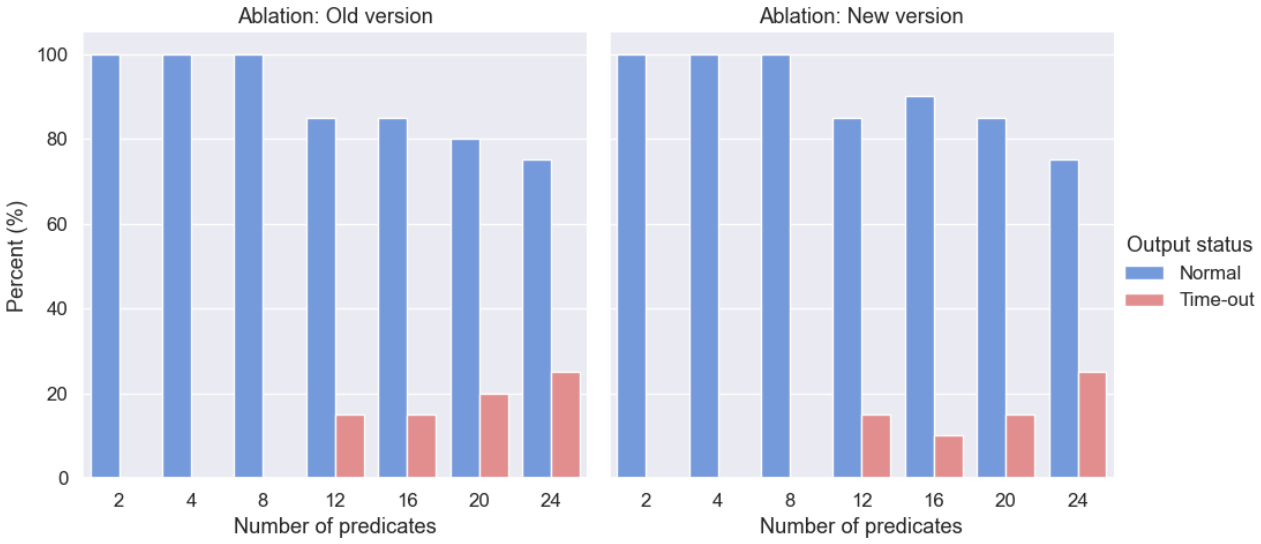
Similar to model_opt, we see that the upper bound on accuracy poses great improvements, though mainly when optimizing for accuracy. We see small but steady improvements with the linearization of $R_{m,j}$, with all ablations that contain this amendment scoring better than the ones without. For the linearization of $H_{g,j}$ we see however that it does not lead to improvement, in fact it only increases the runtime. We therefore conclude that it is not a beneficial amendment, and move forward with the linearization of $R_{m,j}$ and the upper bound on accuracy for order_opt.

5.2. Optimizer complexity

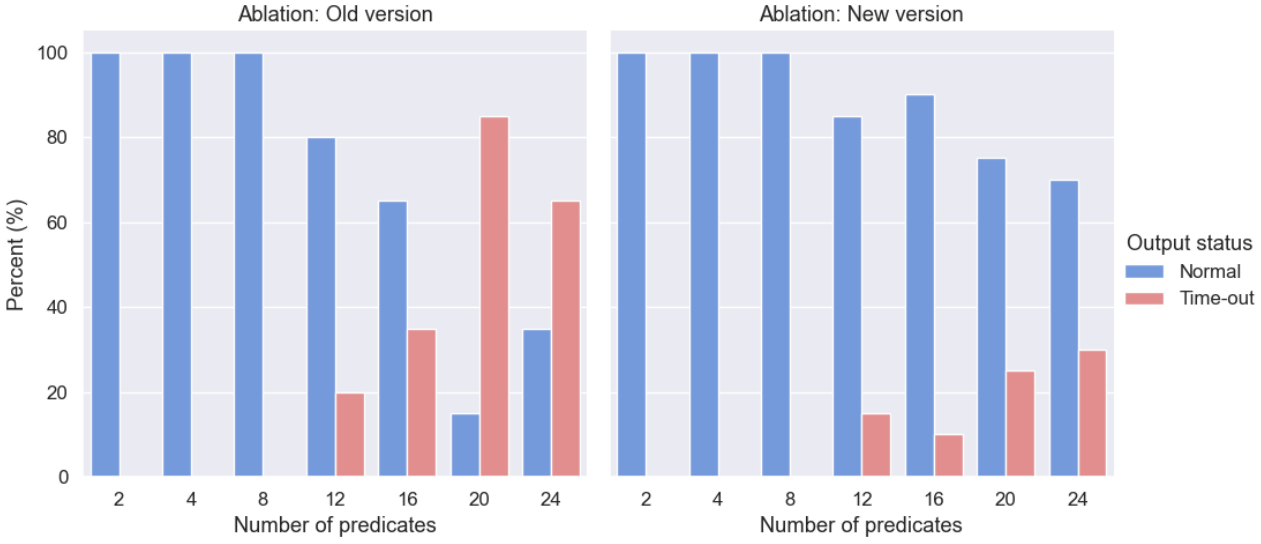
In addition to showing all the amendments improved the optimizer, the complexity of the optimizer with regards to the amount of predicates in a query is also worthy to investigate. The complexity of the original optimizer has already been investigated by Li in the original paper [21]. For this section we are interested in the complexity of the original optimizers versus the versions we propose.

We run the model (optimizing for either accuracy or cost) on a total of 70 queries, varying over the number of predicates. The total list of queries can be found in the appendix. Only two ablations are now considered, the original versions of the optimizers and the versions with all equations amended. After 30 minutes, the output is truncated. This gives somewhat unideal results, but generally we would not be interested anyway in an optimizer that needs that long to compute a solution. For the original versions a bound halfway between the optimal and worst value was chosen. Because we wanted to prioritize the amount of queries over the 'true' runtime of individual queries, we only ran a single trial over every experiment.

First we consider the 'bug report'. Due to working with larger amounts of predicates, a large amount of queries did not finish (dnf).



(a) Model_opt

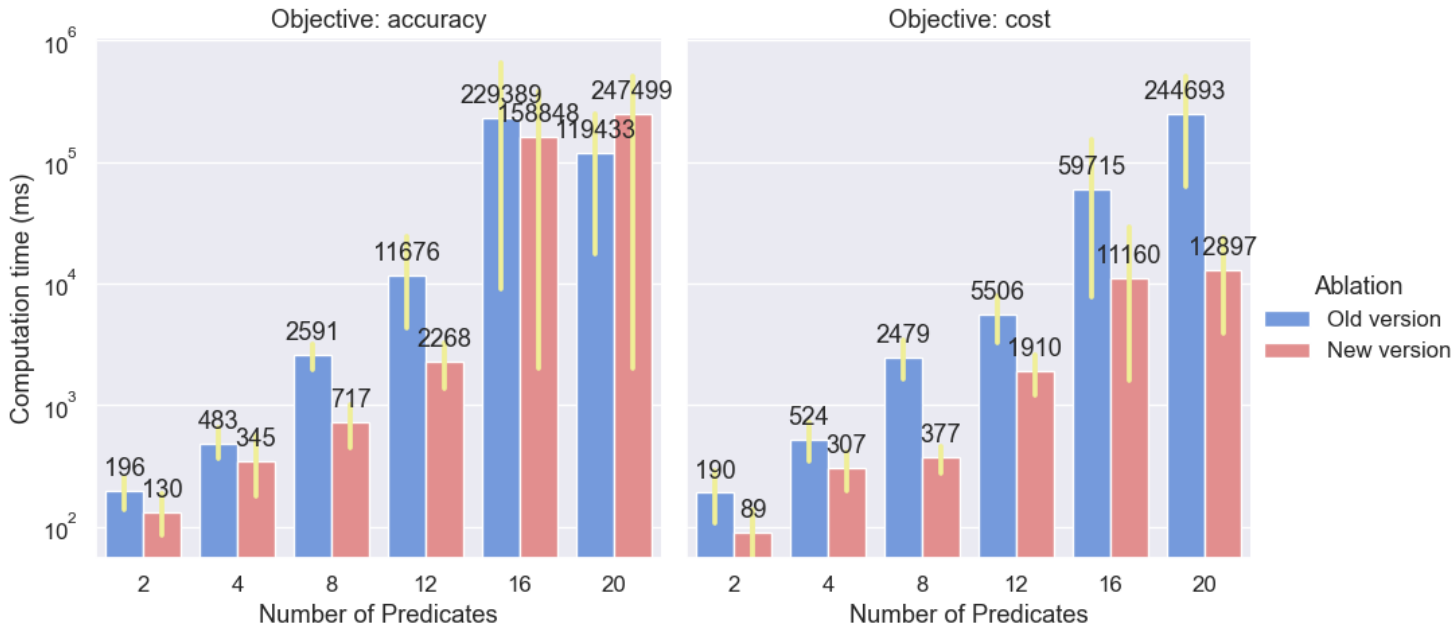


(b) Order_opt

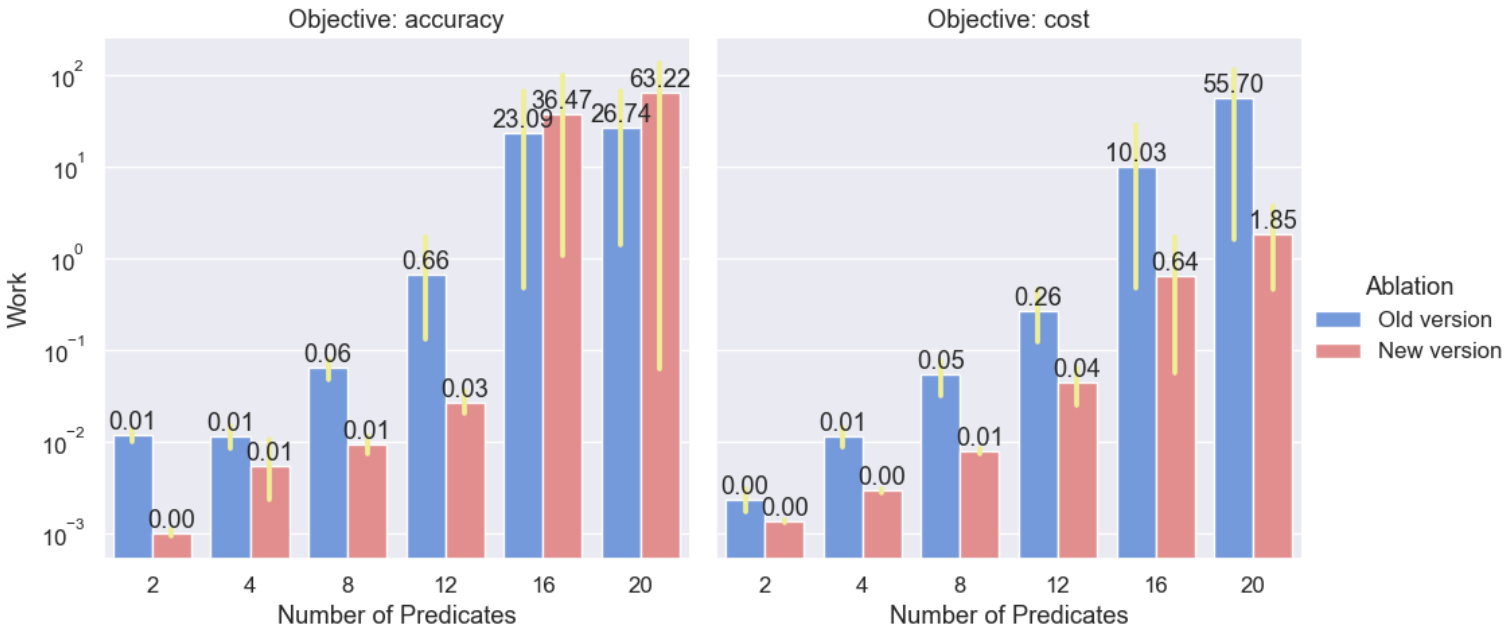
Figure 5.5: Bug report for running the optimizers over queries of a varying amount of predicates.

We see that for model_opt the new version slightly outperforms the original version in terms of time-outs. For order_opt we see a much bigger difference however, with many more queries successfully reaching a solution within half an hour. It is worth mentioning that the results posted in the original paper boast a much lower computation time than achievable in this test setup, with only a few seconds for order_opt for a high amount of predicates. The reason for this discrepancy has not been found.

Next we consider the results for model_opt.



(a) Computation time (in ms)

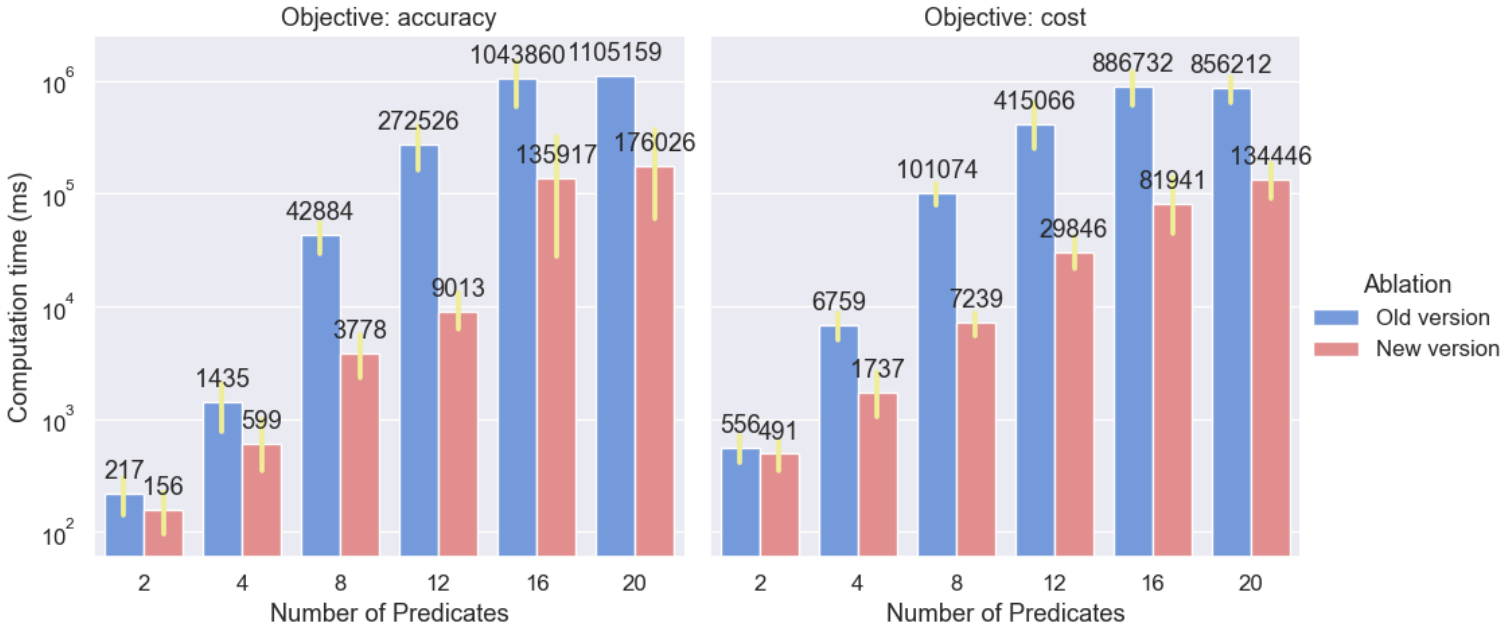


(b) Work

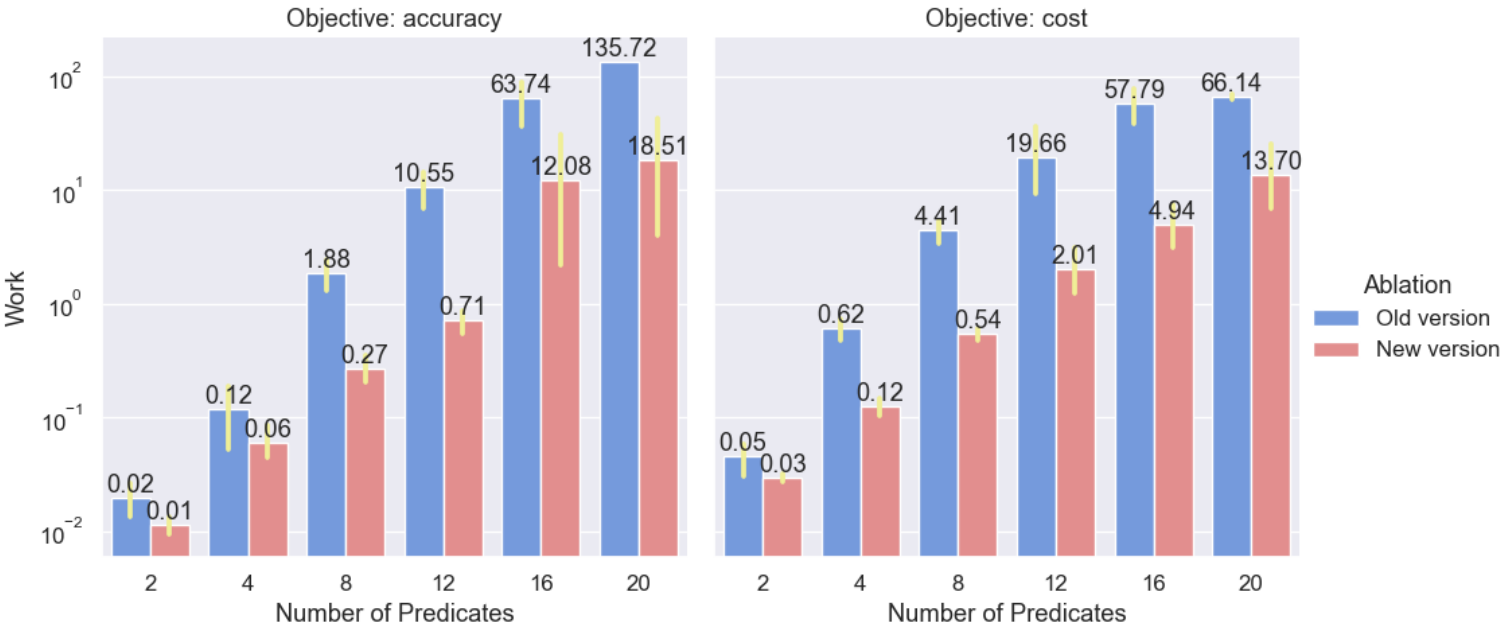
Figure 5.6: Results for the different versions of model_opt

Again we use work as support for the claims we make about computation time. For these results we removed the queries that timed out. This explains why we see inconsistent results: for the old version fewer experiments successfully reached optimal solution within half an hour. The ones that did could be considered to be 'easier' queries, and our new version managed to find a result within less time generally. However, for some of the harder queries the new version also reached a solution, thereby increasing the average. Therefore it is quite hard to make relevant claims about the amount of reduction in computational complexity the new version gives over the old version. This is also because we removed the 24-predicate results, as too many reached a time-out in order to make meaningful conclusions.

We therefore tentatively conclude that for small amounts of predicates our new version outperforms the old version for both accuracy and cost. For larger amount of predicates the differences between both version become very pronounced for cost, but not very much so for accuracy. For order_opt we retrieve similar findings:



(a) Computation time (in ms)



(b) Work

Figure 5.7: Results for the different versions of order_opt

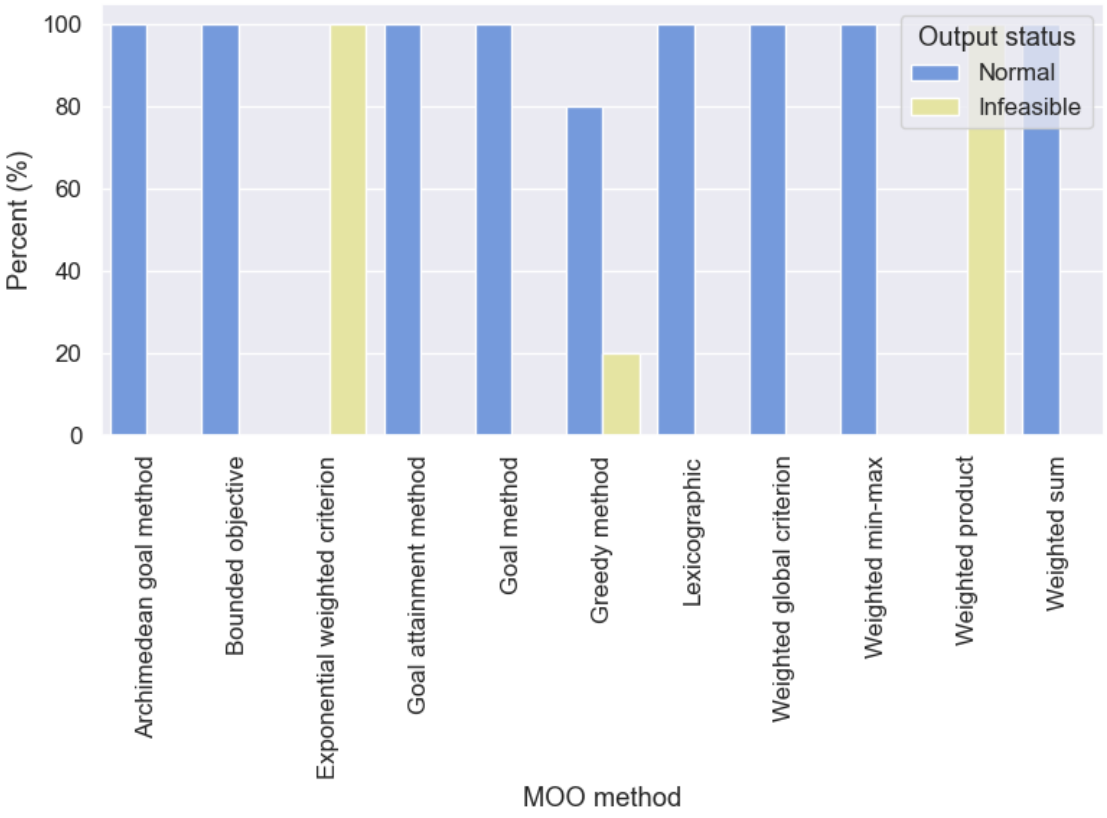
All in all we see a reduction in the overall complexity for the optimizer. Drawing meaningful conclusions about the amount of improvement the new version provides over the old version is still difficult to quantify however.

5.3. MOO method comparison

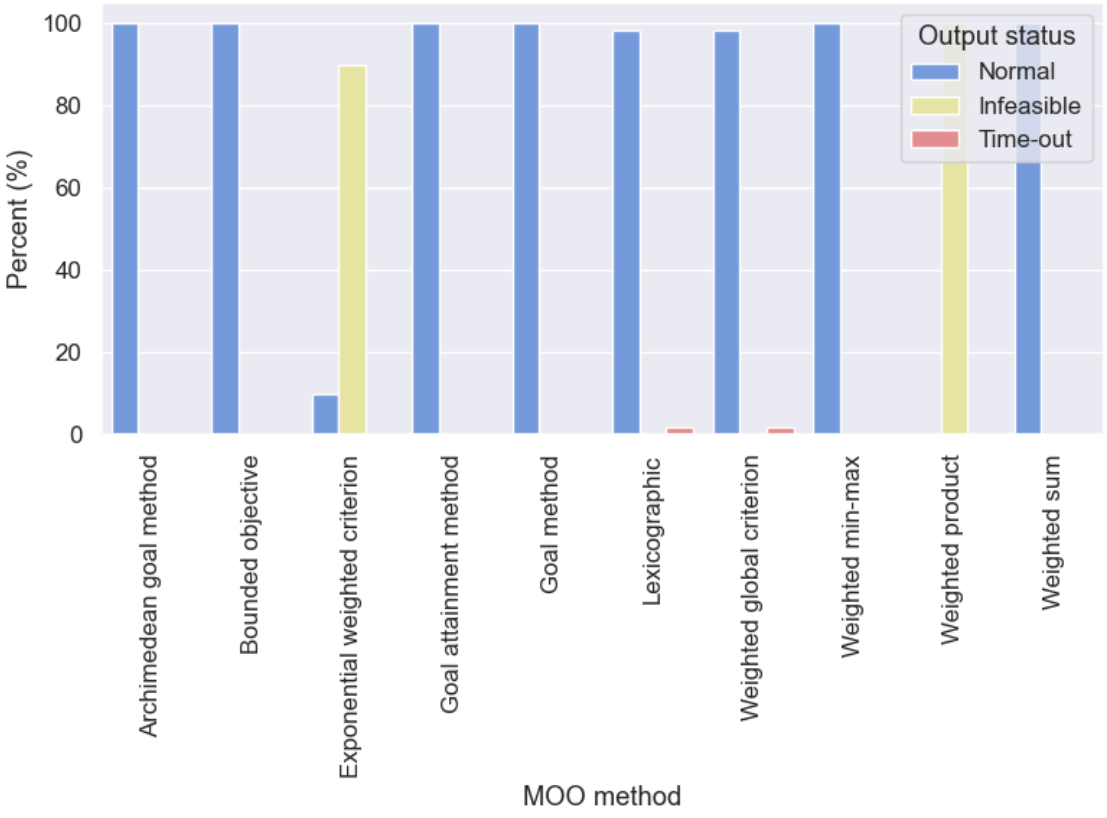
Now that the computational gains of our optimizers have been shown, it is time to investigate the different MOO methods on the optimizer. For this we test a set of 4-predicate queries, as this allows to run more trials within the same amount of time. The query list can be found in the appendix.
For every query we test 6 different preference types:

- accuracy→cost→memory
- accuracy→memory→cost
- cost→accuracy→memory
- cost→memory→accuracy
- memory→accuracy→cost
- memory→cost→accuracy

on every MOO method. This hierarchy allows us to calculate weights using the method specified in section 4.3.3 **No preference types using objectives with equal importance were considered**. For goal methods we used 0 for the most important objective, 0.1 for the second, and 0.2 for the third. This does not necessarily guarantee Pareto optimality, but we wanted to make sure that we were not emulating other methods by setting the utopia point as the goal (see section 4.4.7 for elucidation). For the bounded objective method we used 0 as the lower bound for every objective, 0.25 as the upper bound for the second most important objective, and 0.5 for the least (the most important objective is used to optimize on and does not require bounds). For the methods that use a p-exponent, we set $p = 3$. For every experiment we again record the computation time, work, and optimizer output status. The runtime cutoff is 30 minutes. The total amount of trials we run for every experiment is 10.
First we consider the 'bug reports':



(a) Bug report for model_opt



(b) Bug report for order_opt

Figure 5.8: Bug reports for model_opt and order_opt over several queries for several MOO methods

It is clear that the methods relying on many product variables (exponential weighted criterion and weighted product) are performing not as desired, with most experiments returning infeasibility errors. While a valid solution exists, the amount of product variables likely makes the problem to difficult to solve for Gurobi. The lexicographic method and the weighted global criterion also suffer from timeouts. If we then consider the computation time and work for model_opt (excluding results with error codes):

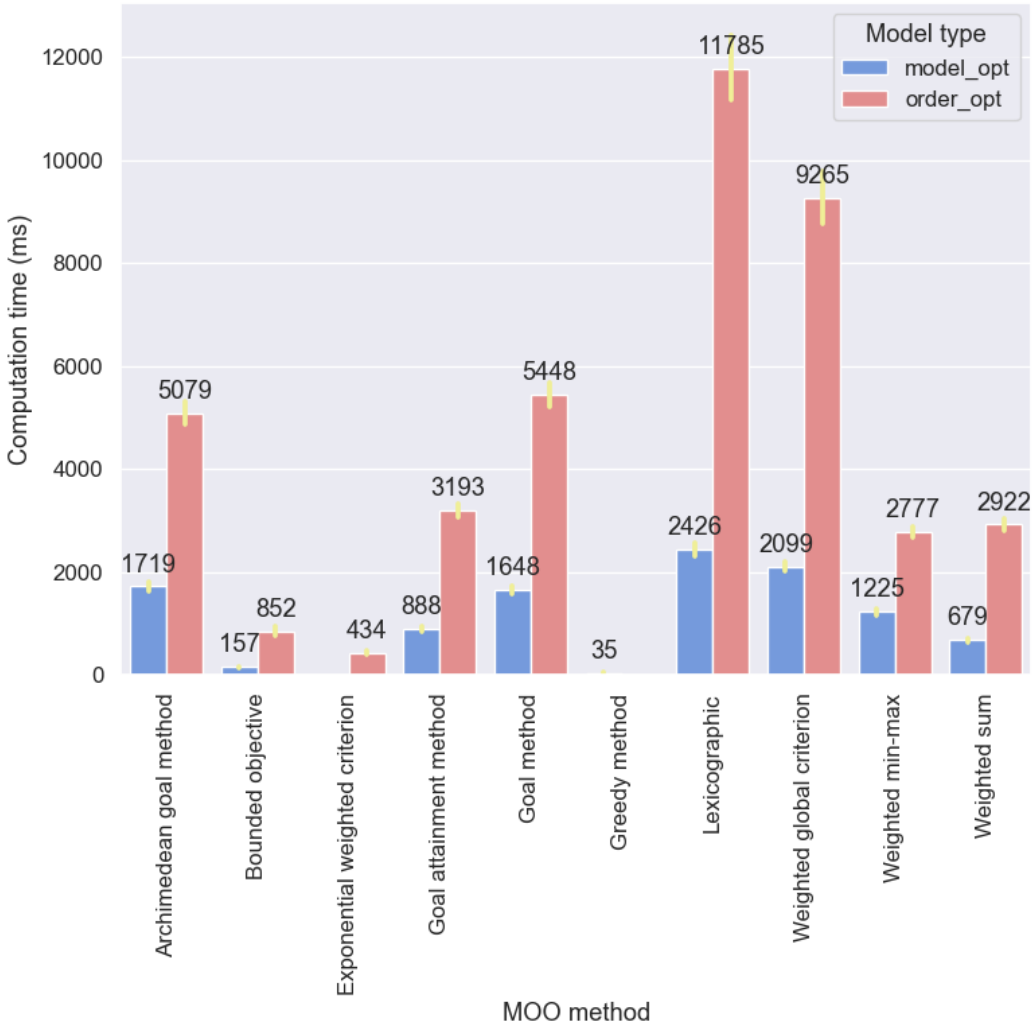


Figure 5.9: Computation time in ms

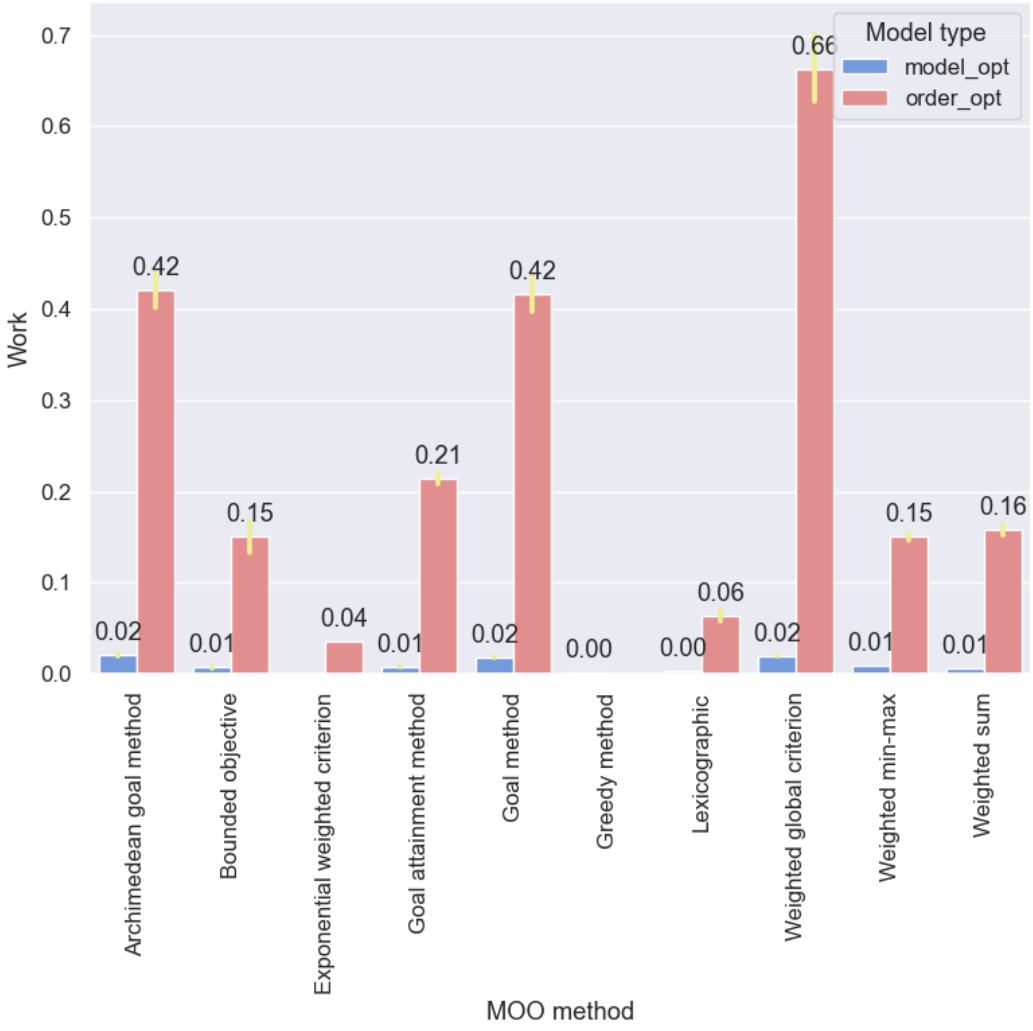


Figure 5.10: Work

We see that most methods are performing somewhat in the range of 0-6000 ms. As the goal of this section is to not only compare MOO methods among each other, but also choose a 'winner' to continue further experiments with, we exclude several methods from our detailed analysis (either due to outliers in computation time or optimization errors):

- Weighted global criterion
- Exponential weighted criterion
- Weighted product
- Lexicographic method

After discarding these methods, we can look at the results in more detail:

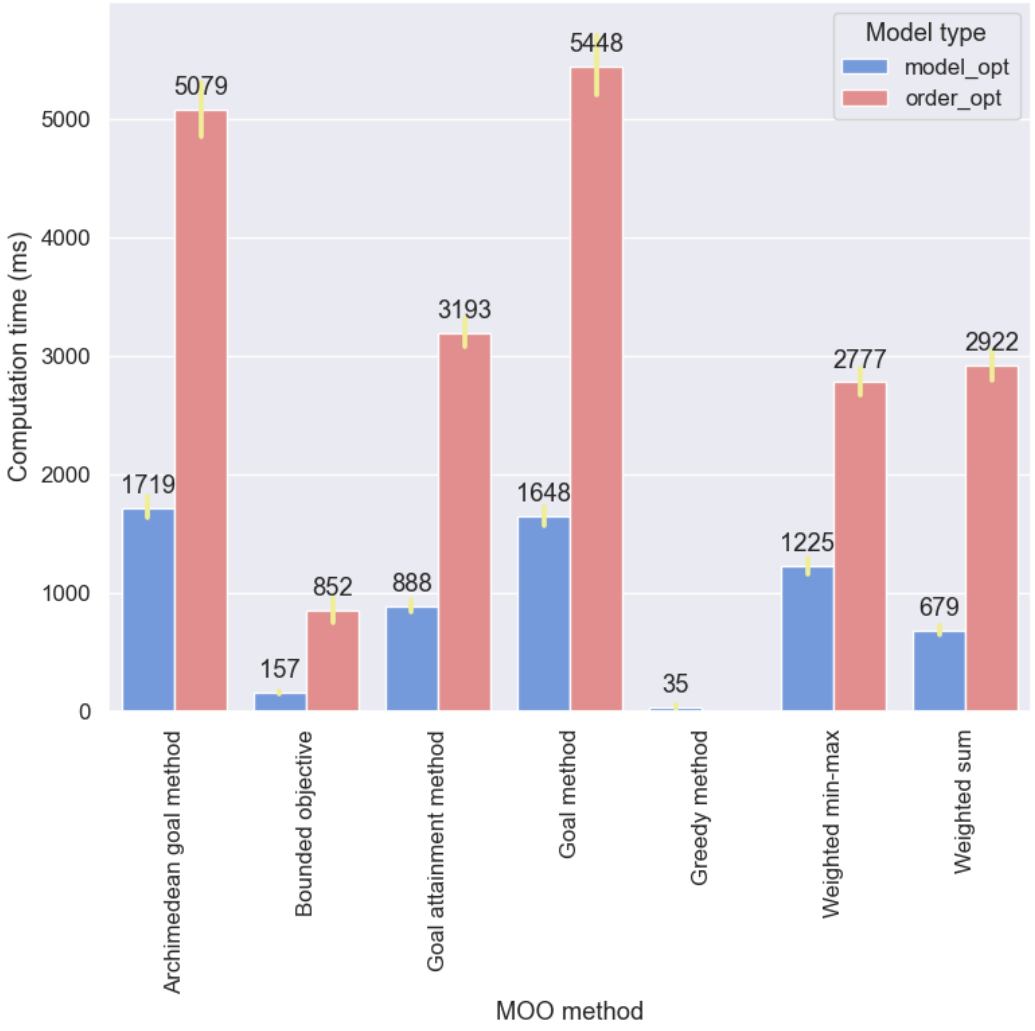


Figure 5.11: Computation time in ms

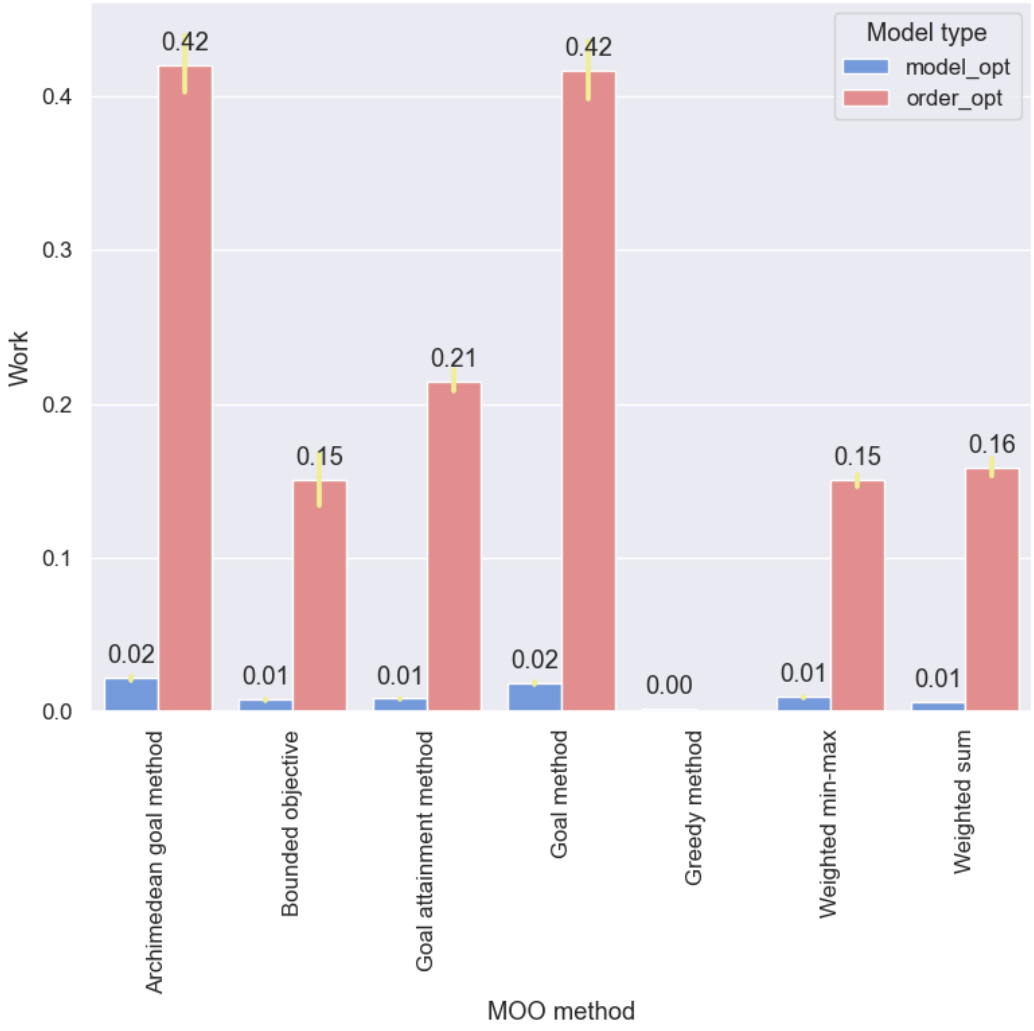


Figure 5.12: Work

The 8 methods left are now compared in more detail on runtime but also on suitability. The main criteria we consider are

- Pareto sufficiency/necessity
- Ability to process preferences
- Optimization time

Archimedean goal method
The Archimedean goal method performs badly in comparison to the other methods in terms of computation time. The possibilities of user input are quite versatile however: the combination of setting a goal with weights allows the user to specify preferences in a more extensive way than other methods. Especially the corollary that a Pareto-optimal goal point will lead to a Pareto optimal solution is a powerful aspect. For less experiences user setting the utopia point as the goal point is a good compromise, even though this method thereby emulates the weighted sum method.

Bounded objective method
The bounded objective method performs very well in terms of computation time, almost competing with the greedy method for model_opt. One issue however remains the lack of Pareto optimality, and the risks with infeasibility. It is up to the user to choose bounds that lead to a solution, and there might be some trial and error involved. Especially when more objectives are involved, simply calculating the greedy solution is not sufficient.

Goal attainment method
The Goal attainment method performs fairly well in terms of computation time. One issue remains with the lack of guarantee of Pareto optimality. Weak Pareto sufficiency is still good, but with the other methods that can guarantee (strong) Pareto optimality with the same possibilities of user input the method is not a great contender.

Goal method
The goal method performs worst in terms of computation time. It is sufficient for Pareto optimality as long as the utopia point is set as the goal. There is however no further opportunity for indication of preferences.

Greedy method
The greedy method performs, like expected, best of all methods in terms of computational speed. The method does not allow for any indication of preference however. It would be possible to include weights, but more sophisticated methods of calculating solutions exist.

Weighted min-max
The weighted min-max method performs well in terms of computation time, with some outliers. The average computation time is fairly low and the standard deviation is small. The biggest problem is however with the lack of Pareto sufficiency, which it cannot guarantee.

Weighted sum
The weighted sum performs well in terms of average computation time. There are a few outliers, but not a lot. It is a powerful method, as it is Pareto sufficient and uses weights as indicators of preference. However, when conventional weight determination methods are insufficient, conveying preference through numerical values is difficult.

Final choice of method
After considering 10 methods, it is possible to make a choice of what is the best method to proceed with for the optimizer. We evaluated all methods on their robustness (i.e. whether every run returned valid results), computation time, Pareto sufficiency, and the possibility for the user to customize. The **Archimedean goal method** returns valid results across all runs, has acceptable computation time, can guarantee Pareto optimality, but also allows for intuitive customization using both weights and goals. It is therefore the best choice for this application. The second choice would be the weighted sum method, but the Archimedean goal method emulates the weighted sum method anyway when the utopia point is set as the goal.

5.4. NLP experiments

Like mentioned in the introduction, this work also concerns itself with practical applications of the query optimizer. We have demonstrated the capabilities of our optimizer in a synthetic setting (only using model zoo metadata), but we have not yet investigated the performance of our query plans on actual data. While Li's original paper [21] already proved the merit of using her optimizer over naive method, we are curious whether our MOO-oriented query plans also balance objectives than naive methods (i.e. greedy MOO). For the application domain of this work, we look at the field of NLP. This is due to the natural nuances of language that are less prominent in object detection. For example, the meaning of a sentence can drastically change depending on word ordering, the sentiment with which it is said, and other subtleties. In short, the possibility of adding a second predicate to a simple boolean predicate, thereby making it a more complex query (e.g. ”toxic and ”negative sentiment” versus solely ”toxic”). Even though we do not rewrite popular NLP tasks to CNF or DNF queries, we have illustrated our motivations to explore NLP more extensively than covered in the original work by Li [21].

We therefore generate a small NLP model zoo of 2 tasks with several predicates. The reason why we generate the model zoo ourselves is because most widely available models classify all predicates within a task. Therefore, to make for an interesting optimization problem, a lot of tasks need to be considered. It can be quite a lot of effort to compile such a zoo, and therefore we opted for choosing to train our own models and compare with our own baselines.

The tasks we are considering are

1. Toxicity classification
2. Sentiment analysis

For toxicity classification we use the jigsaw toxic comment classification dataset [16]. This dataset contains a training dataset of about 150000 Wikipedia comments, that are labelled as ”toxic”, ”severe_toxic”, ”threat”, ”identity_hate”, ”obscene”, and ”insult”. These labels are not exclusive, making it possible to formulate queries on just this dataset. The Kaggle competition also gives a (partially) labeled test set, with about 15000 labeled items.

For sentiment analysis we derive ground truth on the Jigsaw dataset using the model with highest accuracy on huggingface[3]. We treat the sentiment analysis as a classification problem, where a sentiment is either ”negative”, ”neutral”, or ”positive.

On this dataset two types of models are trained:

- Logistic regression
- Long-short term memory neural network (LSTM)

The models are taken from the Kaggle code page[4], and reformulated them to classify sentiment analysis or toxicity classification, with some variations in hyperparameters for the LSTM networks. For implementation details, we refer to the supplementary GitHub[5]. After training the models on 60% of the training data, the rest of the data is used as a testing set. Both accuracy and f1-scores are tracked to allow for optimization over different accuracy metrics, but we optimize over f1-score, as there is greater variance in this value. The full model zoo can be found in the appendix.

We generate a number of queries and generate query plans, which we execute on the test set. This is not ideal, as the scores in the model zoo have also been derived using this testset. However, we are not trying to show the performance of the model, but the efficacy of our query plans. We consider a total of 40 queries, varying over 2,4,6 and 8 predicates. The query list can be found in the appendix. For every query we generate a query plan using order_opt with the Archimedean goal method, with the utopia point as the goal. We use equal weights for all objectives, in order to emulate the preference pattern of greedy MOO. Obviously the power of our method is that we can process user preferences, but in this experiment we are trying to show that our MOO method outperforms naive methods. Therefore we also generate a query plan using greedy MOO as a comparison baseline. Greedy MOO uses the default ordering of predicates for execution, with the model selection defined in section 4.3.4.1.

While the query plans were generated using the hardware setup mentioned in section 5.1, for running the query plans on actual data we used one of the TU Delft clusters. The cluster boasts 256 GB RAM, a 24 core CPU, and an Nvidia A40. The model zoo was also generated with these specifications.

For the evaluation of results, it is important to keep a few things in mind. Broadly speaking, results in academic research should show that your method is improving on a certain metric of the current state of the art. In machine learning this could mean that your model has higher accuracy, or that it achieves similar accuracy as other methods in a shorter amount of time. However, MOO is a fuzzy field: we cannot irrefutably say that one solution is better than the other, as this is preference dependent. If we were to order results as one being better than the other, we would have to define a utility function or something similar (and many MOO methods already use a utility function to optimize anyway). The only way we could confidently say that our optimizer is better than greedy MOO, would be if our optimizer consistently gives a better value for every objective. This would be analogous to saying that our optimizer generates Pareto optimal solutions (as long as a Pareto optimal goal is set) and greedy MOO does not, which we already both know to be true from literature[25] and our proof (section 5.

[3] https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment?text=I+like+you.+I+love+you

[4] Logistic regression: https://www.kaggle.com/code/tunguz/logistic-regression-with-words-and-char-n-grams
LSTM: https://www.kaggle.com/code/sbongo/for-beginners-tackling-toxic-using-keras

[5] https://github.com/marietteschonfeld/MILQO

Keeping this in mind, we consider the results. For these figures we visualize both the average of the obtained objectives (so what was measured, not predicted by our optimizers) and the actual values in bar charts:.

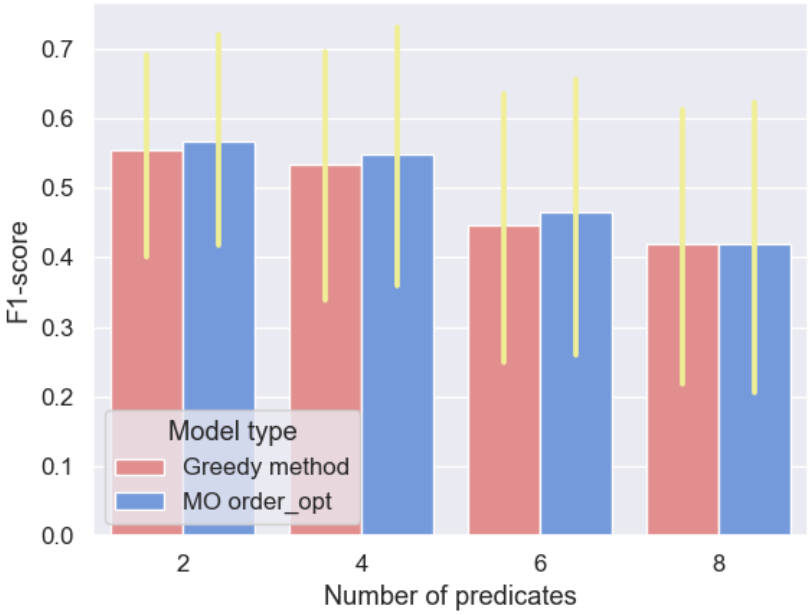


Figure 5.13: Average f1-score using the MO order_opt query plan versus the greedy method

We see that the MO order_opt optimizer gives slightly higher accuracy than the greedy method. If we look at individual query results:

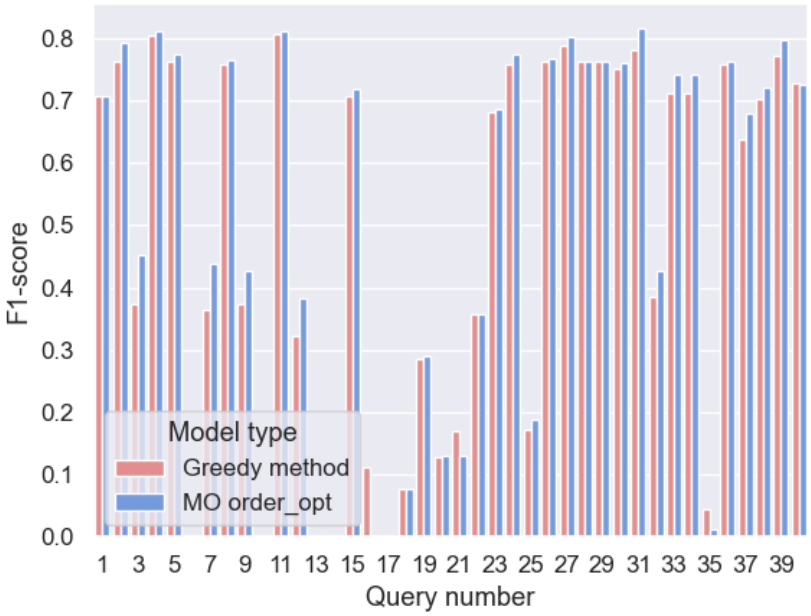


Figure 5.14: f1-score of all queries using the MO order_opt query plan versus the greedy method

We can see that there are a handful of queries with zero f1-score, but that for most queries our optimizer finds a query plan with higher f1-score. Next we consider the total evaluation time:

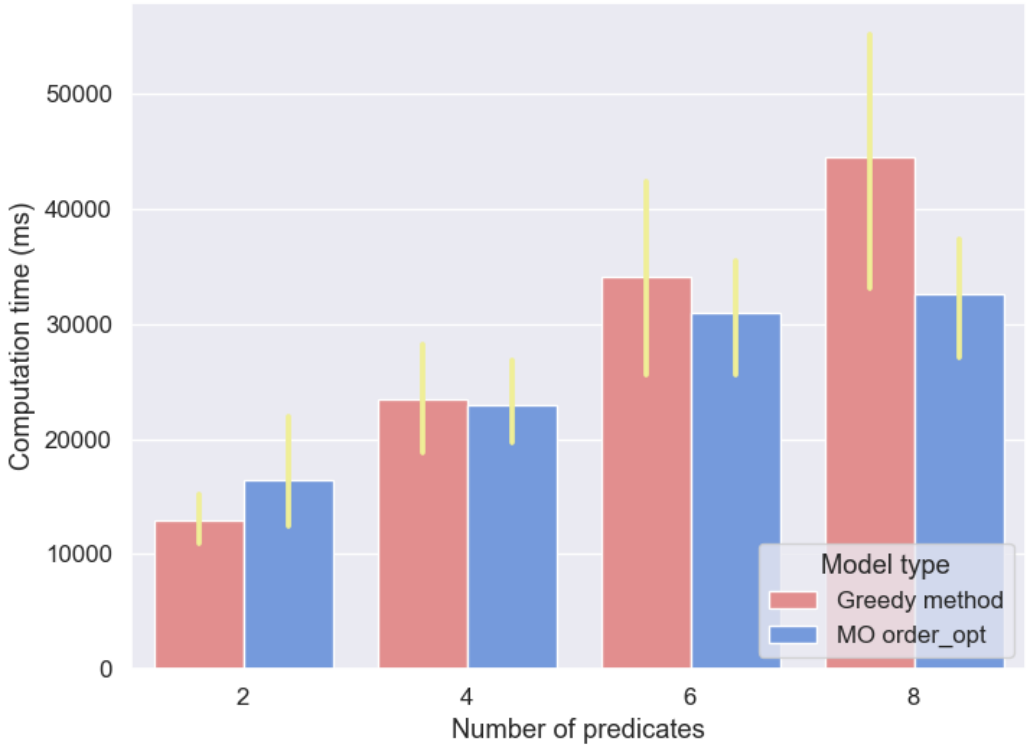


Figure 5.15: Average of computational cost of all queries using the MO order_opt query plan versus the greedy method

We see that for computation time that the greedy algorithm performs slightly better for the smaller amounts of predicates. However, when the amount of predicates increases, the MO order_opt query plan outperforms the greedy method. This is confirmed by the individual results:

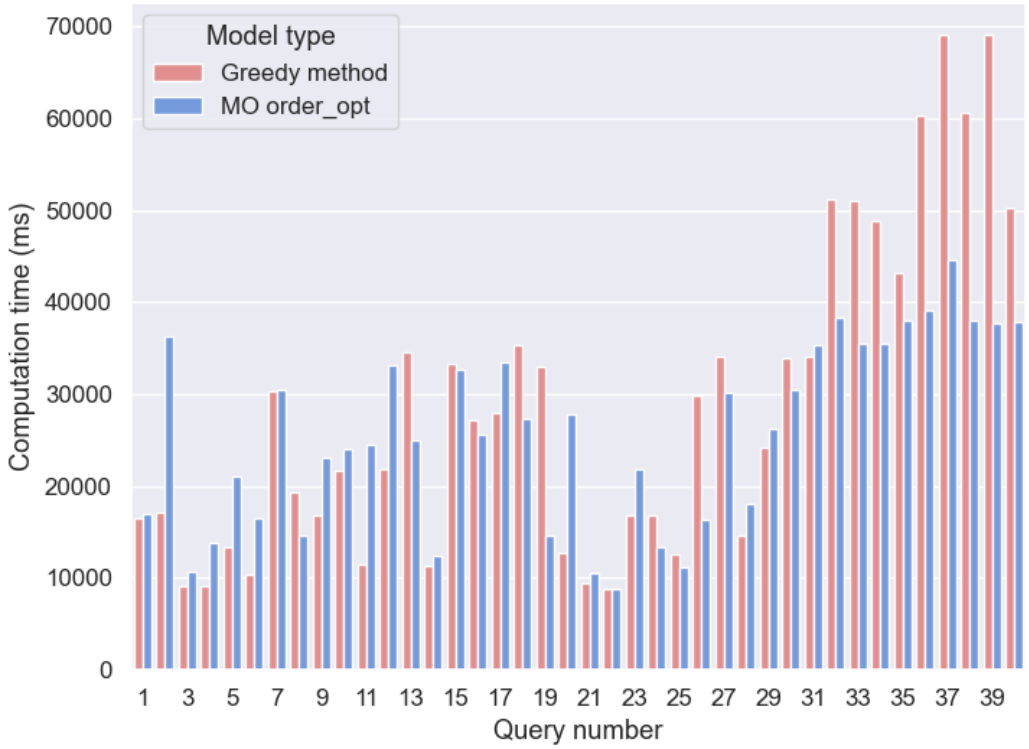


Figure 5.16: Computation time of all queries using the MO order_opt query plan versus the greedy method

Finally we look at the results for memory footprint:

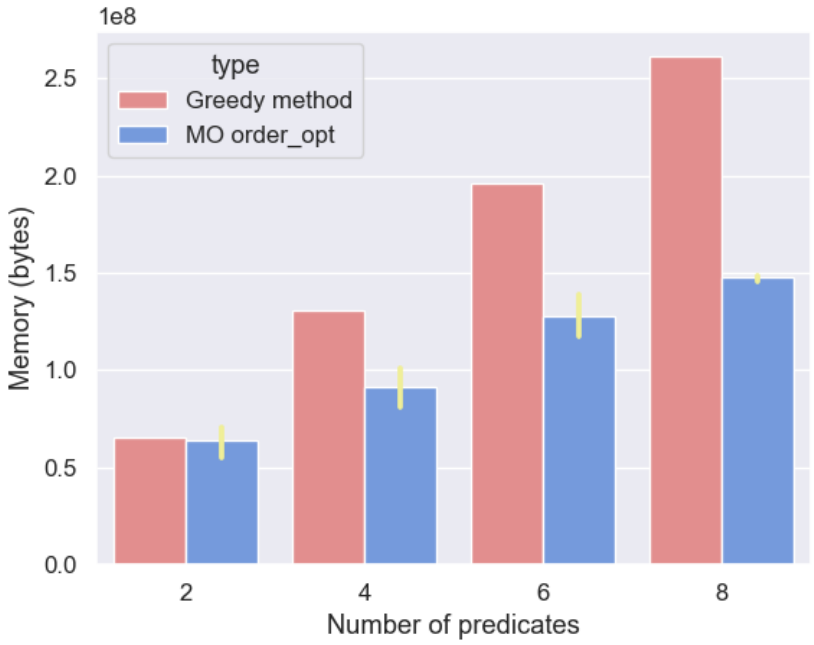


Figure 5.17: Average of memory footprint of all query plans using the MOO query plan versus the greedy method

For the memory footprint we see consistent improvements of MOO over the greedy method. This is confirmed by the individual results:

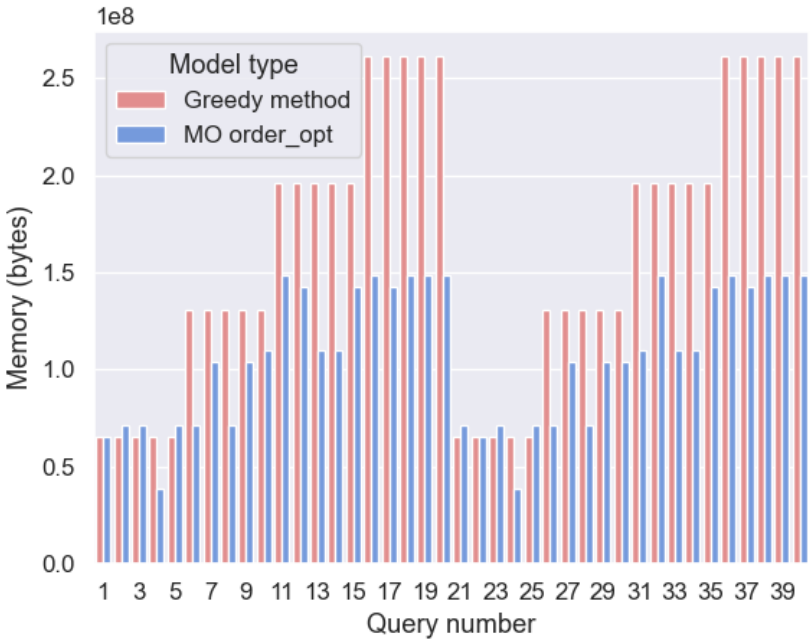


Figure 5.18: Memory of all queries using the MO order_opt query plan versus the greedy method

We therefore tentatively conclude that MO order_opt generates better query plans, as it manages to generate query plans with comparable/slightly better f1-score, less cost as the number of predicates increases, and lower memory footprint. It is worth mentioning that our model zoo is not of great quality: it contains only two types of models, and the execution time of the LSTM models is about 10 times that of the LR models. Therefore, the greedy query plans greatly prioritize the LR models, while our optimizer prefers to balance memory footprint and accuracy by choosing the LSTM models, which can classify a larger amount of predicates with slightly higher accuracy. In a more balanced model zoo the benefit of our optimizer might be more apparent.

6

Conclusion

6.1. Summary

We have proposed an optimizer that can generate query plans for ML inference query plans which are optimized for multiple objectives. Out optimizer improves on the computational complexity of the original optimizer while generating higher-quality solutions, though the percentual gains remain difficult to determine.

We have added memory as an objective to the optimizer, and compared several MOO-methods on their suitability for the optimizer. The chosen method allows for user customization in two ways, while generating Pareto-optimal solutions.

The resulting optimizer manages to find higher-quality query plans than naive methods for real NLP data, thereby showing its capabilities in juggling objectives.

6.2. Evaluation of research questions

We discuss the initial research questions and whether we have found a satisfying answer for them.

What is the current standard for deriving machine learning inference query plans?
The current standard for finding ML inference query plans using model zoos can be found in a paper by Li et al [21]. This paper proposes two MIP optimizers, an optimal-selection optimizer and an order-optimal optimizer.

How do you define an optimizer for such a query that takes multiple objectives into account?
Due to Li using MIP optimizers, there are many MOO methods that can be implemented, by simply amending the goal function. We have compared multiple methods and have chosen the Archimedean goal method, as it allows for customization using both numerical weights and goal setting. The method has sufficiently competitive runtime in comparison to other methods, but can guarantee Pareto optimality if the goal is beyond the Paretio frontier.

Is there room for adding other objectives?
Due to original optimizer being an MIP, adding a new objective is simple once the equations of modelling the problem are derived. We therefore added the memory footprint of the models in the query plan as an objective.

How does this optimizer perform against the current state of the art?
Due to the lack of comparable algorithms, we propose a naive greedy method that simulates MOO to compare our optimizer to. Our optimizer finds query plans that are emanating the same preference profile while improving objective values, especially for larger amounts of predicates.

How does this optimizer perform on NLP tasks?

The optimizer can find query plans for common NLP tasks in a reasonable amount of time, efficiently juggling objectives. The generated query plans manage to find the data that matches the query with higher accuracy and lower computation time than naive methods.

6.3. Considerations and recommendations

6.3.1. Approximation schemes

One of the prominent problems with our proposed optimizer remains the issue of runtime, especially for queries for larger amounts of predicates. In an exploratory phase, we tested summing accuracies of predicates, rather than multiplying them. This seemed to greatly reduce runtime, in line with our expectations of product variables negatively impacting runtime. The documentation of Gurobi specifies approximation schemes being used for non-convex problems, but the inner workings of this remain unknown. However, the approximation schemes that Gurobi uses might be too sophisticated for our needs. Using a simpler approximation scheme might reduce runtime. In addition to runtime, there are also other reasons why we might prefer a simpler method. One big aspect of MOO is Pareto optimality. However, when two solutions have extremely similar accuracy (like 0.98712 and 0.98713) both could be considered to be Pareto optimal. For our case, we might only be interested in 4 decimal points. Therefore we would like our optimizer to mark these two solutions as having equal accuracy. Doing so would reduce the size of the Pareto frontier, discarding a lot of solutions we would consider to be of equal quality.

6.3.2. Heuristics

One other opportunity to reducing runtime could be that of using heuristics over exact solving methods. A heuristic is a method that prioritizes finding a sufficiently good solution within reasonable time over finding the exact optimal solution. Heuristics lend themselves especially well to MIPs, and many methods use neighbor-based search (also known as local search) strategies, such as Tabu search [10] or simulated annealing [32]. Because these methods make the impact of product variables void, they could be of use to our optimizer.

6.3.3. Memory as an objective

In section 4.3.2 we added the memory or storage footprint of models as a third objective. This objective is not necessarily the most interesting to optimize for however. Storage space is relatively cheap in comparison to other computational resources, especially for the scale we are needing it. There are however two other ways we can utilize memory:

1. The amount of intermediate data that needs to be worked with
2. The cost of unloading a large model

For this work there wasn't enough time to investigate this matter in more detail. Therefore we add our conjectures to this chapter for further research.

Size of intermediate data

The idea of using intermediate data storage as an objective for query optimization is not novel. In fact, one of the papers mentioned in the related work section, namely that of Immanuel Trummer and Christoph Koch [30], also utilizes this concept. With intermediate memory we mean the amount of data that is moved down the 'tree'. As we want to filter data that does not fit the query we can just not inference the data which we already know will never fit the query. For example, when looking for yellow Volkswagen Polos, if a quarter of the pictures have a yellow object after running the first model, we can discard 75% of the data, and our intermediate storage after the first step is only 25% of what we started with. We only have to run the 'Polo' model on the remaining 25% of data.

For this objective almost all of the building blocks have already been defined. The amount of data that is used per step is already encapsulated within S_j^J. If we define the total amount of bytes of data we would like to model inference one as Δ, we can then calculate the total memory as

$$f_{mem}(q) = \sum_{m \in M} B_m D_m + \sum_{j \in J} S_j^J \Delta \tag{6.1}$$

One issue which prevented us from investigating this matter properly was the mismatch between memory/storage footprint and internal memory. The storage footprint of a model is quite easy to calculate: the amount of bytes that a file takes on disk is a builtin-function in Python. The internal memory of an object is a lot more complicated to calculate, and is heavily dependent on the programming language in which it is used. To figure out a way to unify storage footprint, a model's internal memory usage and intermediate data size proved to be a problem more complicated than initially thought. Investigating this could be an interesting research alley, especially for edge (i.e. mobile) applications.

Cost of unloading a model
We already mentioned the relatively small issue of memory footprint. Moving a model from non-volatile disk space to RAM in order to model inference is an expensive operation however [27]. From the aforementioned book, we want to highlight the staggering difference between accessing time of magnetic disk (HDD) versus RAM memory: about 400000 times as long. The difference between flash semiconductor memory (SSD) and RAM is less big (about 100 times in good circumstances), but still considerable. In short, we want to avoid loading a model to RAM unless it has a lot of merit. Especially for users working with HDD memory, the time penalty for model loading versus model execution could be much more interesting to optimize for. There are two methods to go about this.

Relative cost
The accuracy of the original cost model of Li's model has not been investigated in detail. If our cost model was perfect, we would not need to do a lot of work, as we can just add the loading time of a model to the execution time and let the optimizer find the solution with smallest cost. If we do not have a very accurate predictor of cost, we can also 'lead' the optimizer to solutions that we pre-emptively know to be of merit. Therefore we can think about the **relative** cost of unloading a model. If we are going to have to load an expensive model, we should at least run it on as much data as possible. This is correlated with selectivity, but is not necessarily data-dependent. Therefore we can formulate an objective that is not order-dependent.

First we define the amount of milliseconds needed to transfer a single byte from disk to RAM as δ. It would be entirely possible to estimate this time per model like we do with execution cost, but that would not give the user the opportunity to fill in a value for δ that is dependent on their hardware setup, which is exactly what we would like to enable them to do.
The cost of unloading a model $m \in M$ is then equal to

$$\delta D_m$$

We want to make this time relative to the amount of data we hope to be inferencing on using this model. This is correlated with the predicate selectivity S_p^P, but somewhat inversely. For execution cost, we want to avoid running expensive models on large amounts of data. In this case we wish to only use expensive models if we are expecting to have to evaluate a large amount of data. Therefore we use

$$1 - S_p^P$$

and sum this over all predicates that model m wants to answer:

$$\sum_{p \in P} (1 - S_p^P) X_{m,p}$$

And then the relative cost of unloading a model m is

$$\delta D_m \sum_{p \in P} (1 - S_p^P) X_{m,p}$$

If we now add this to the original cost functions we derive for model_opt:

$$f_{cost}(q) = \sum_{m \in M} \left(B_m C_m + (\sum_{p \in P} (1 - S_p^P) X_{m,p}) \delta D_m \right) \tag{6.2}$$

and for order_opt we get

$$f_{cost}(q) = \sum_{m \in M} \left(\max_{j \in J} R_{m,j} + \delta D_m \sum_{p \in P} (1 - S_p^P) X_{m,p} \right) \tag{6.3}$$

This amended cost model might move the optimizer to solutions that have a lower total model unloading time, in case when the user's δ value is particularly high in comparison to the model execution time.

A new cost model
Like mentioned before, the ideas above do not have to be used when we have an accurate cost model for execution time. In that case we could just add the model unloading time to the execution cost, and the optimizer will naturally find the solution with the best total execution time. But evaluating how accurate our cost model is would not be the easiest task however. Questions that need to be answered include (but are not limited to):

- How accurate is the projected execution cost on actual data?
- Which factors contribute to large inaccuracies (such as number of predicates, group size, data size, etc)?
- How do we fix these issues?

In short, there are many more issues to be investigated around this model.

6.4. Epilogue

"Wow detective, that turned out to be a much more complex problem than I initially thought." "Indeed constable, it is a shame that it took us almost 10 months, but I guess we were quite thorough in our research." "I suppose our findings are not that relevant anymore for the hit-and-run case." "No, but we found answers that are relevant for all problems of this form. I hope that we, or some other person, can use our methodology for good." "I'm a little bit sad that we are done though. It was only starting to get interesting. Shall we have a crack at the implementation of Tabu search?" "Grab your computer constable, we have work to do".

Bibliography

[1] R. Anand, D. Aggarwal, and V. Kumar, “A comparative analysis of optimization solvers,” *Journal of Statistics and Management Systems*, vol. 20, no. 4, pp. 623–635, 2017.

[2] M. R. Anderson, M. Cafarella, G. Ros, and T. F. Wenisch, “Physical representation-based predicate optimization for a visual analytics database,” *arXiv e-prints*, arXiv–1806, 2018.

[3] M. Asghari, A. M. Fathollahi-Fard, S. Mirzapour Al-e-hashem, and M. A. Dulebenets, “Transformation and linearization techniques in optimization: A state-of-the-art survey,” *Mathematics*, vol. 10, no. 2, p. 283, 2022.

[4] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, 4. Springer, 2006, vol. 4.

[5] A. Charnes and W. W. Cooper, “Goal programming and multiple objective optimizations: Part 1,” *European journal of operational research*, vol. 1, no. 1, pp. 39–54, 1977.

[6] E. F. Codd, “A relational model of data for large shared data banks,” in *Software pioneers*, Springer, 2002, pp. 263–294.

[7] G. B. Dantzig, “Programming in a linear structure,” in *Bulletin of the American Mathematical Society*, AMER MATHEMATICAL SOC 201 CHARLES ST, PROVIDENCE, RI 02940-2213, vol. 54, 1948, pp. 1074–1074.

[8] R. Elmasri, S. B. Navathe, R. Elmasri, and S. Navathe, *Fundamentals of Database Systems</Title*. Springer, 2000.

[9] F. Gembicki and Y. Haimes, “Approach to performance and sensitivity multiobjective optimization: The goal attainment method,” *IEEE Transactions on Automatic control*, vol. 20, no. 6, pp. 769–771, 1975.

[10] F. Glover, “Tabu search: A tutorial,” *Interfaces*, vol. 20, no. 4, pp. 74–94, 1990.

[11] D. Gómez and A. Rojas, “An empirical overview of the no free lunch theorem and its effect on real-world machine learning classification,” *Neural computation*, vol. 28, no. 1, pp. 216–228, 2016.

[12] Z. Gu, E. Rothberg, and R. Bixby, *Gurobi optimizer*, `https://gurobi.com/`, Online. Accessed on August 17th, 2022, 2009.

[13] N. Gunantara, “A review of multi-objective optimization: Methods and its applications,” *Cogent Engineering*, vol. 5, no. 1, p. 1 502 242, 2018.

[14] Gurobi, *Gurobi 9.0 delivers improved performance and major new features*, `https://www.gurobi.com/news/gurobi-9-0-delivers-improved-performance-and-major-new-features/#:~:text=Gurobi%209.0%20delivers%20breakthrough%20new,programming%20(MIQP)%20problem%20types.`, Online. Accessed on August 17th, 2022, 2020.

[15] HuggingFace, *Huggingface*, `https://huggingface.co/`, Online. Accessed August 16th 2022.

[16] G. Jigsaw, *Jigsaw toxic comment classification challenge*, `https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge`, Online. Last accessed on October 3rd, 2022, 2017.

[17] D. Kang, P. Bailis, and M. Zaharia, “Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics,” *arXiv preprint arXiv:1805.01046*, 2018.

[18] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, “Noscope: Optimizing neural network queries over video at scale,” *arXiv preprint arXiv:1703.02529*, 2017.

[19] K. Karanasos, M. Interlandi, D. Xin, *et al.*, “Extending relational query processing with ml inference,” *arXiv preprint arXiv:1911.00231*, 2019.

[20] Z. Li, *Optimizing-ml-query*, `https://github.com/zLizy/Optimizing-ML-Query`, 2022.

[21] Z. Li, M. Schönfeld, W. Sun, *et al.*, “Optimizing inference queries with model repositories,” *Not yet published*, 2022.

[22] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, “Microsoft coco: Common objects in context,” in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[23] Y. Lu, A. Chowdhery, S. Kandula, and S. Chaudhuri, “Accelerating machine learning inference with probabilistic predicates,” in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1493–1508.

[24] R. T. Marler and J. S. Arora, “Survey of multi-objective optimization methods for engineering,” *Structural and multidisciplinary optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[25] K. Miettinen, *Nonlinear multiobjective optimization*. Springer Science & Business Media, 2012, vol. 12.

[26] S. Papadomanolakis and A. Ailamaki, “An integer linear programming approach to database design,” in *2007 IEEE 23rd International Conference on Data Engineering Workshop*, IEEE, 2007, pp. 442–449.

[27] D. A. Patterson and J. L. Hennessy, *Computer organization and design ARM edition: the hardware software interface*. Morgan kaufmann, 2016, p. 378.

[28] R. T. Rockafellar, “Lagrange multipliers and optimality,” *SIAM review*, vol. 35, no. 2, pp. 183–238, 1993.

[29] TensorFlow, *Tensorflow hub*, `https://www.tensorflow.org/hub`, Online. Accessed August 16th 2022.

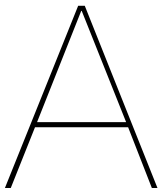
[30] I. Trummer and C. Koch, “Multi-objective parametric query optimization,” *The VLDB Journal*, vol. 26, no. 1, pp. 107–124, 2017.

[31] ——, “Solving the join ordering problem via mixed integer linear programming,” in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1025–1040.

[32] P. J. Van Laarhoven and E. H. Aarts, “Simulated annealing,” in *Simulated annealing: Theory and applications*, Springer, 1987, pp. 7–15.

[33] D. Xu, I. E. Yen, J. Zhao, and Z. Xiao, “Rethinking network pruning–under the pre-train and fine-tune paradigm,” *arXiv preprint arXiv:2104.08682*, 2021.

[34] K. P. Yoon and C.-L. Hwang, *Multiple attribute decision making: an introduction*. Sage publications, 1995.



Additional results

Like mentioned in section 5.1, looking at results of individual queries is more enlightening than the average. Therefore we include these figures here.

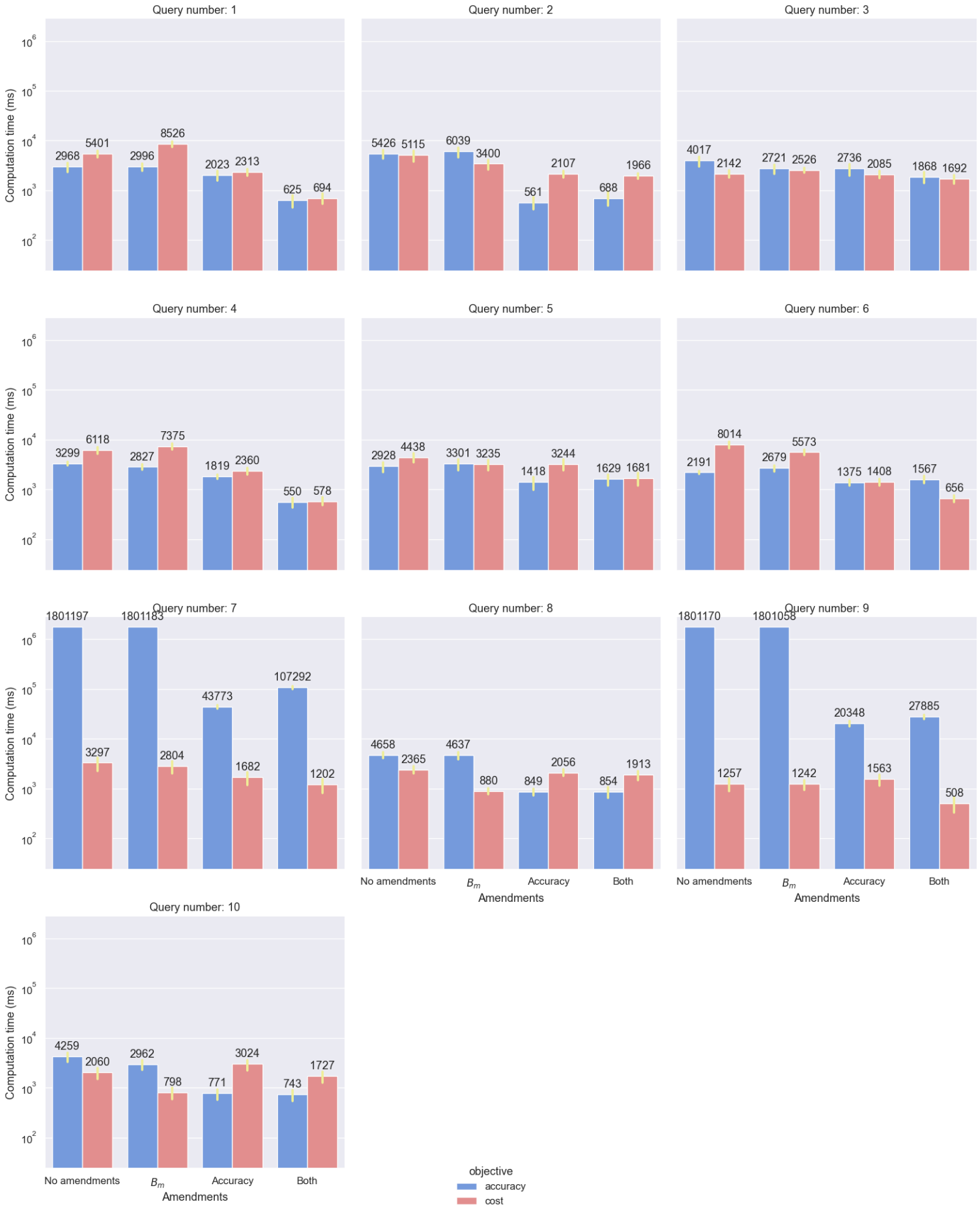


Figure A.1: Computation time for model_opt over several queries

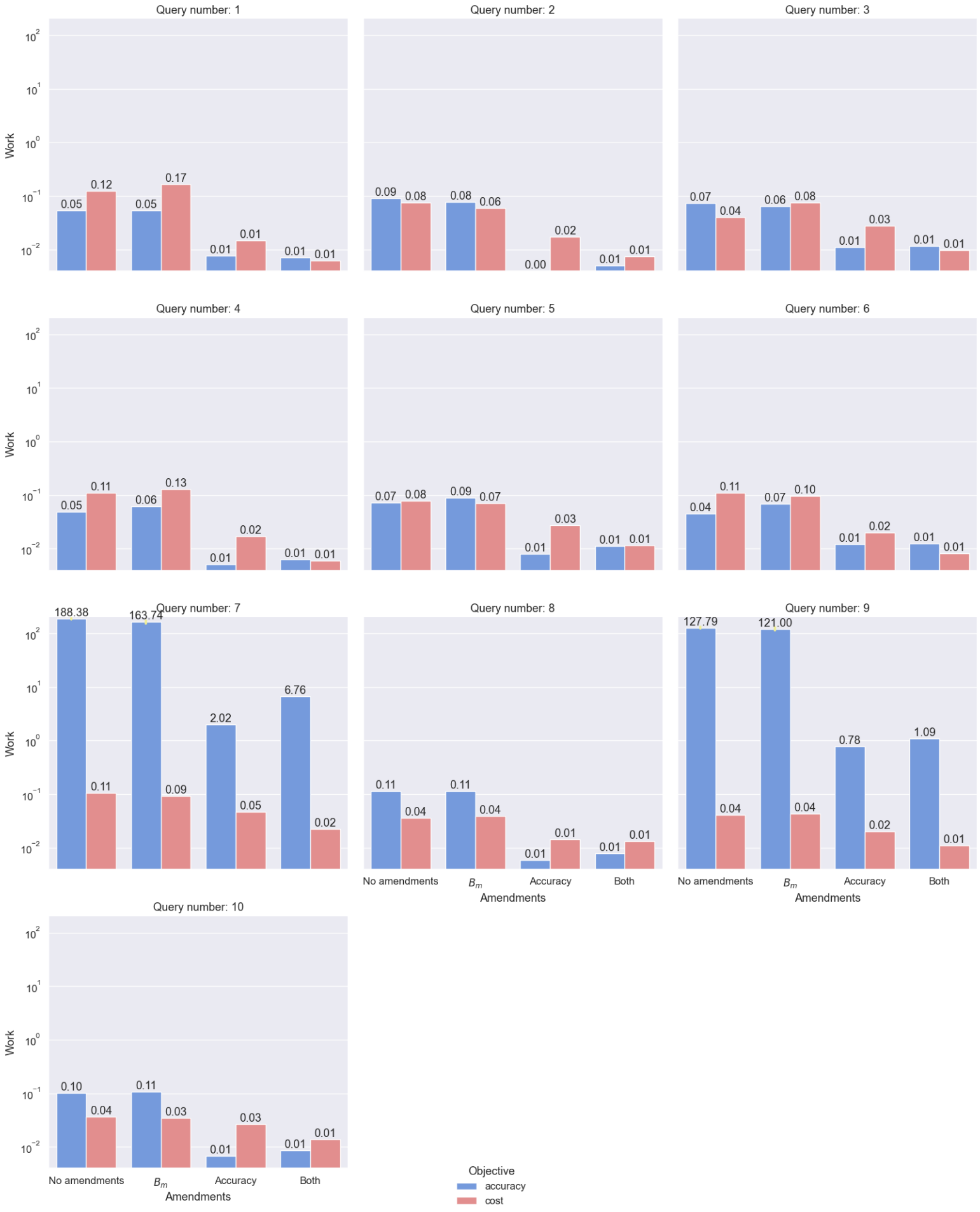


Figure A.2: Work for model_opt over several queries

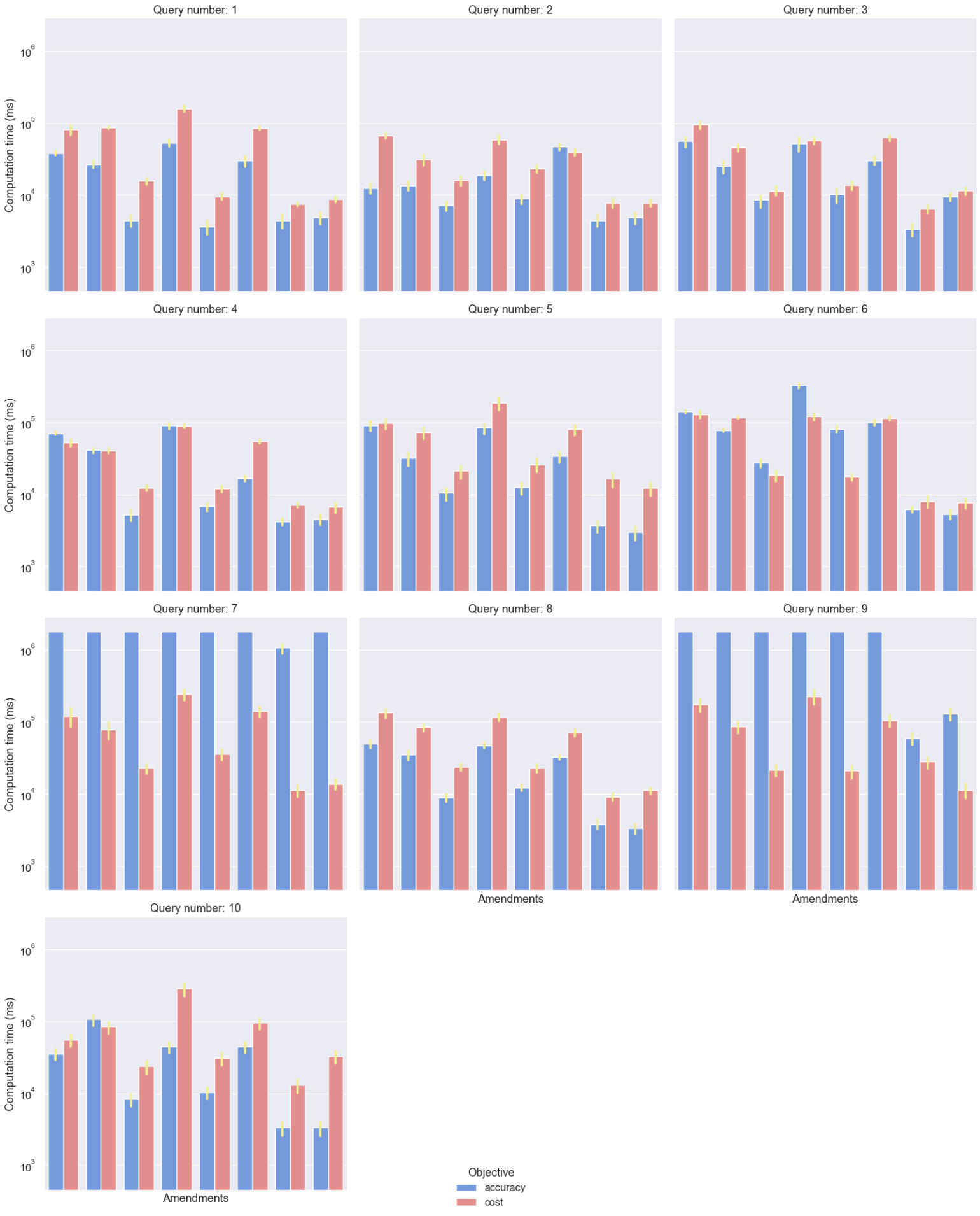


Figure A.3: Computation time for model_opt over several queries

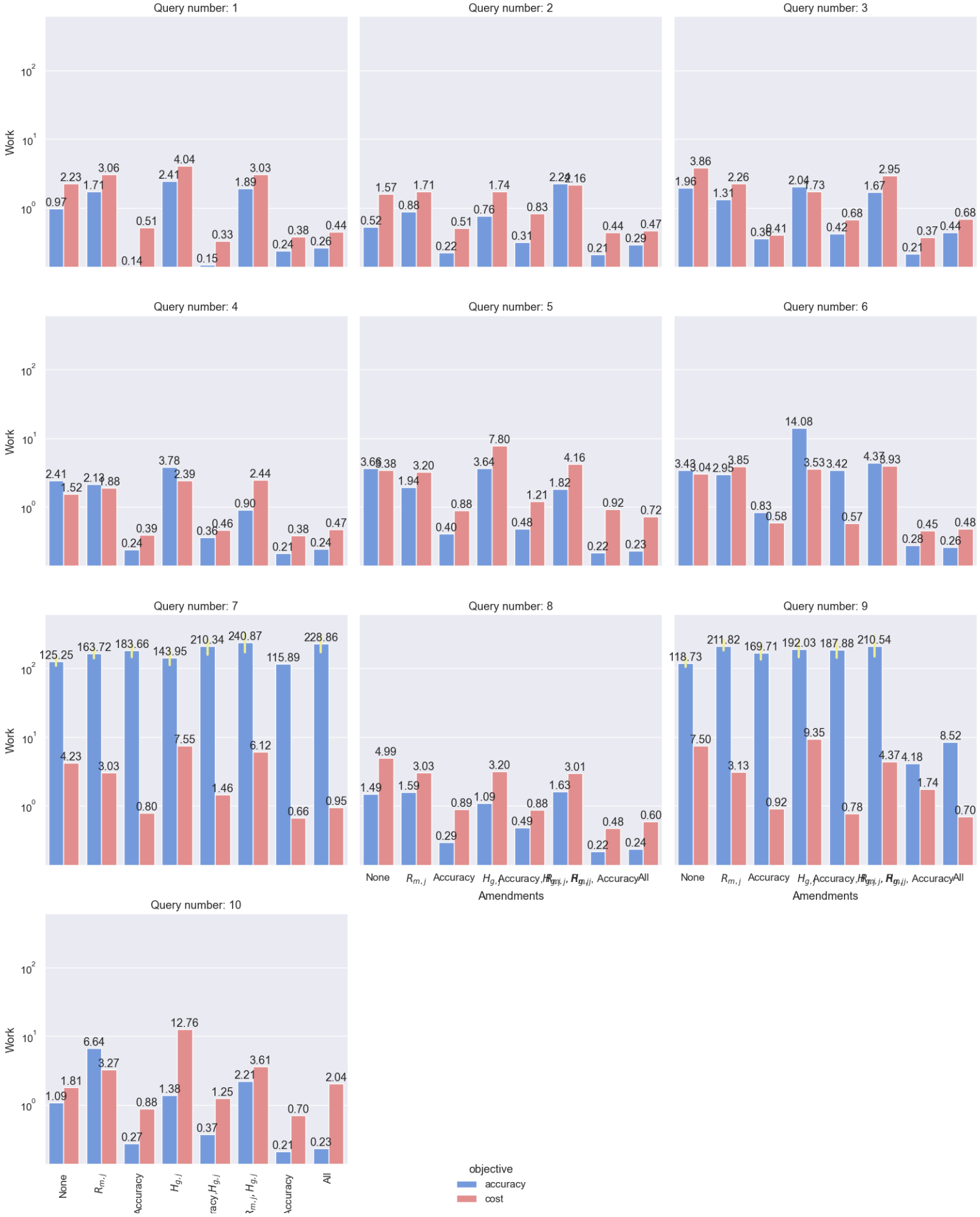


Figure A.4: Work for model_opt over several queries

B

Query list

B.1. 8 predicate query list

The following queries were used for the experiments in section 5.1:

1. $(bottle \vee keyboard) \wedge (zebra) \wedge (umbrella \vee motorcycle) \wedge (airplane \vee pizza \vee stop\_sign)$
2. $(pizza \vee skateboard) \wedge (frisbee) \wedge (toothbrush \vee hot\_dog) \wedge (dining\_table) \wedge (fork) \wedge (sports\_ball)$
3. $(scissors \vee truck) \wedge (frisbee \vee skis \vee mouse) \wedge (teddy\_bear \vee remote) \wedge (bus)$
4. $(stop\_sign \vee chair \vee boat) \wedge (carrot \vee skateboard) \wedge (bear) \wedge (pizza) \wedge (toothbrush)$
5. $(bench \vee scissors) \wedge (bed \vee dog) \wedge (pizza) \wedge (sheep \vee dining\_table \vee tv)$
6. $(bottle \wedge keyboard) \vee (zebra) \vee (umbrella \wedge motorcycle) \vee (airplane \wedge pizza \wedge stop\_sign)$
7. $(pizza \wedge skateboard) \vee (frisbee) \vee (toothbrush \wedge hot\_dog) \vee (dining\_table) \vee (fork) \vee (sports\_ball)$
8. $(scissors \wedge truck) \vee (frisbee \wedge skis \wedge mouse) \vee (teddy\_bear \wedge remote) \vee (bus)$
9. $(stop\_sign \wedge chair \wedge boat) \vee (carrot \wedge skateboard) \vee (bear) \vee (pizza) \vee (toothbrush)$
10. $(bench \wedge scissors) \vee (bed \wedge dog) \vee (pizza) \vee (sheep \wedge dining\_table \wedge tv)$

B.2. Queries with increasing amount of predicates

The following queries were used for the experiments in section 5.2:

1. $(laptop) \wedge (surfboard)$
2. $(toaster) \wedge (zebra)$
3. $(microwave \vee fork)$
4. $(cell\_phone) \wedge (book)$
5. $(bus \vee spoon)$
6. $(elephant \vee surfboard \vee cat) \wedge (apple)$
7. $(microwave \vee dining\_table \vee teddy\_bear) \wedge (fire\_hydrant)$
8. $(cup \vee apple) \wedge (horse \vee banana)$
9. $(elephant \vee person \vee skis) \wedge (parking\_meter)$
10. $(baseball\_glove) \wedge (fire\_hydrant \vee tv \vee surfboard)$
11. $(donut \vee clock \vee bench) \wedge (boat \vee surfboard) \wedge (laptop \vee skis) \wedge (scissors)$
12. $(teddy\_bear \vee frisbee) \wedge (giraffe \vee cow \vee handbag) \wedge (broccoli \vee apple \vee parking\_meter)$
13. $(motorcycle) \wedge (bowl \vee sink \vee oven) \wedge (train \vee traffic\_light) \wedge (cell\_phone \vee toothbrush)$
14. $(car \vee hot\_dog \vee pizza) \wedge (airplane) \wedge (surfboard \vee chair) \wedge (cow \vee teddy\_bear)$
15. $(truck \vee sports\_ball \vee toaster) \wedge (skateboard \vee teddy\_bear) \wedge (bus \vee laptop \vee cat)$
16. $(refrigerator \vee scissors) \wedge (kite \vee spoon \vee frisbee) \wedge (tv) \wedge (baseball\_bat) \wedge (traffic\_light \vee keyboard \vee mouse) \wedge (baseball\_glove \vee person)$

17. $(vase) \wedge (book) \wedge (scissors \vee frisbee \vee bed) \wedge (handbag \vee orange) \wedge (sink \vee donut) \wedge (oven \vee dining\_table \vee surfboard)$
18. $(wine\_glass) \wedge (baseball\_glove \vee kite \vee bus) \wedge (bowl \vee skateboard \vee zebra) \wedge (parking\_meter \vee horse \vee book) \wedge (train \vee surfboard)$
19. $(banana) \wedge (donut) \wedge (baseball\_glove) \wedge (vase \vee remote) \wedge (couch) \wedge (bench \vee boat) \wedge (traffic\_light \vee giraffe) \wedge (umbrella \vee bowl)$
20. $(donut \vee suitcase \vee motorcycle) \wedge (cup \vee car \vee orange) \wedge (spoon) \wedge (sink) \wedge (horse) \wedge (sandwich \vee cell\_phone) \wedge (couch)$
21. $(remote \vee sports\_ball) \wedge (skis \vee kite \vee vase) \wedge (sheep \vee cat \vee bench) \wedge (toothbrush) \wedge (umbrella \vee bottle \vee suitcase) \wedge (truck \vee snowboard) \wedge (toilet) \wedge (bed)$
22. $(boat \vee chair \vee laptop) \wedge (sports\_ball \vee spoon \vee kite) \wedge (apple \vee frisbee) \wedge (baseball\_glove \vee cow \vee bowl) \wedge (zebra \vee dog \vee truck) \wedge (bench) \wedge (skis)$
23. $(fire\_hydrant \vee dog \vee zebra) \wedge (skateboard \vee microwave \vee book) \wedge (toilet) \wedge (motorcycle) \wedge (banana \vee vase) \wedge (orange \vee knife) \wedge (cat \vee bottle) \wedge (wine\_glass \vee cup)$
24. $(spoon \vee cup \vee cow) \wedge (fire\_hydrant) \wedge (teddy\_bear) \wedge (giraffe) \wedge (donut \vee dog) \wedge (airplane) \wedge (surfboard \vee chair) \wedge (skis \vee laptop \vee bear) \wedge (sports\_ball \vee bicycle)$
25. $(bird) \wedge (broccoli) \wedge (bowl) \wedge (sheep) \wedge (couch \vee elephant \vee scissors) \wedge (snowboard \vee motorcycle) \wedge (person \vee dining\_table) \wedge (surfboard \vee baseball\_glove) \wedge (teddy\_bear) \wedge (mouse \vee knife)$
26. $(suitcase \vee knife) \wedge (vase \vee car) \wedge (laptop \vee giraffe) \wedge (cell\_phone \vee donut) \wedge (airplane \vee remote \vee dining\_table) \wedge (apple) \wedge (spoon \vee boat) \wedge (tv \vee sandwich \vee carrot) \wedge (stop\_sign \vee orange) \wedge (bed)$
27. $(scissors \vee kite \vee stop\_sign) \wedge (oven \vee cell\_phone \vee wine\_glass) \wedge (donut \vee clock) \wedge (zebra \vee carrot \vee skis) \wedge (teddy\_bear \vee sports\_ball \vee frisbee) \wedge (snowboard \vee train) \wedge (toothbrush \vee laptop \vee umbrella) \wedge (apple)$
28. $(laptop) \wedge (frisbee \vee sink) \wedge (pizza \vee banana \vee teddy\_bear) \wedge (refrigerator \vee toaster \vee bench) \wedge (boat) \wedge (mouse \vee tennis\_racket \vee traffic\_light) \wedge (sports\_ball \vee knife) \wedge (motorcycle) \wedge (bus) \wedge (baseball\_glove \vee skateboard \vee giraffe)$
29. $(sandwich) \wedge (wine\_glass \vee sink \vee person) \wedge (skateboard \vee suitcase) \wedge (dining\_table) \wedge (toaster \vee car) \wedge (clock \vee toothbrush) \wedge (oven \vee carrot) \wedge (skis \vee fork) \wedge (bed) \wedge (horse \vee sports\_ball) \wedge (pizza \vee bear)$
30. $(frisbee) \wedge (couch \vee donut) \wedge (fork \vee microwave) \wedge (tennis\_racket) \wedge (oven \vee backpack) \wedge (toaster \vee bear) \wedge (train) \wedge (orange \vee spoon \vee bench) \wedge (dining\_table) \wedge (bed) \wedge (tv \vee motorcycle \vee horse) \wedge (handbag)$
31. $(stop\_sign) \wedge (person \vee bear) \wedge (chair) \wedge (orange) \wedge (sink \vee cow) \wedge (skateboard \vee toilet) \wedge (bowl \vee frisbee \vee skis) \wedge (spoon \vee fork) \wedge (kite) \wedge (bench \vee truck) \wedge (bird) \wedge (donut \vee laptop) \wedge (tv) \wedge (umbrella \vee refrigerator \vee toothbrush)$
32. $(bottle \vee knife \vee tennis\_racket) \wedge (mouse \vee baseball\_bat) \wedge (toilet) \wedge (couch) \wedge (oven) \wedge (skis) \wedge (cat) \wedge (refrigerator \vee horse) \wedge (cell\_phone \vee clock \vee cup) \wedge (skateboard \vee backpack \vee handbag) \wedge (cake) \wedge (snowboard) \wedge (stop\_sign \vee bus \vee laptop) \wedge (bicycle)$
33. $(sink \vee fork) \wedge (suitcase \vee toaster \vee scissors) \wedge (train) \wedge (snowboard \vee potted\_plant) \wedge (teddy\_bear) \wedge (toilet) \wedge (wine\_glass \vee parking\_meter \vee vase) \wedge (apple \vee carrot) \wedge (handbag \vee sports\_ball \vee toothbrush) \wedge (sandwich \vee surfboard) \wedge (bed) \wedge (person \vee baseball\_bat) \wedge (cup)$
34. $(mouse) \wedge (vase \vee carrot) \wedge (bear \vee tennis\_racket \vee skis) \wedge (clock \vee spoon \vee oven) \wedge (banana \vee scissors) \wedge (sheep \vee potted\_plant) \wedge (pizza \vee bicycle) \wedge (surfboard \vee broccoli) \wedge (wine\_glass \vee baseball\_bat) \wedge (tie \vee cat) \wedge (teddy\_bear \vee sports\_ball \vee bottle)$
35. $(truck \vee baseball\_bat) \wedge (cat \vee cup) \wedge (stop\_sign \vee parking\_meter \vee surfboard) \wedge (remote \vee laptop \vee elephant) \wedge (tie \vee bench \vee car) \wedge (sink) \wedge (backpack \vee bowl \vee couch) \wedge (dog) \wedge (traffic\_light \vee motorcycle) \wedge (sports\_ball \vee toilet) \wedge (sheep \vee bottle)$
36. $(laptop) \vee (surfboard)$
37. $(toaster) \vee (zebra)$
38. $(microwave \wedge fork)$
39. $(cell\_phone) \vee (book)$

40. $(bus \wedge spoon)$
41. $(elephant \wedge surfboard \wedge cat) \vee (apple)$
42. $(microwave \wedge dining\_table \wedge teddy\_bear) \vee (fire\_hydrant)$
43. $(cup \wedge apple) \vee (horse \wedge banana)$
44. $(elephant \wedge person \wedge skis) \vee (parking\_meter)$
45. $(baseball\_glove) \vee (fire\_hydrant \wedge tv \wedge surfboard)$
46. $(donut \wedge clock \wedge bench) \vee (boat \wedge surfboard) \vee (laptop \wedge skis) \vee (scissors)$
47. $(teddy\_bear \wedge frisbee) \vee (giraffe \wedge cow \wedge handbag) \vee (broccoli \wedge apple \wedge parking\_meter)$
48. $(motorcycle) \vee (bowl \wedge sink \wedge oven) \vee (train \wedge traffic\_light) \vee (cell\_phone \wedge toothbrush)$
49. $(car \wedge hot\_dog \wedge pizza) \vee (airplane) \vee (surfboard \wedge chair) \vee (cow \wedge teddy\_bear)$
50. $(truck \wedge sports\_ball \wedge toaster) \vee (skateboard \wedge teddy\_bear) \vee (bus \wedge laptop \wedge cat)$
51. $(refrigerator \wedge scissors) \vee (kite \wedge spoon \wedge frisbee) \vee (tv) \vee (baseball\_bat) \vee (traffic\_light \wedge keyboard \wedge mouse) \vee (baseball\_glove \wedge person)$
52. $(vase) \vee (book) \vee (scissors \wedge frisbee \wedge bed) \vee (handbag \wedge orange) \vee (sink \wedge donut) \vee (oven \wedge dining\_table \wedge surfboard)$
53. $(wine\_glass) \vee (baseball\_glove \wedge kite \wedge bus) \vee (bowl \wedge skateboard \wedge zebra) \vee (parking\_meter \wedge horse \wedge book) \vee (train \wedge surfboard)$
54. $(banana) \vee (donut) \vee (baseball\_glove) \vee (vase \wedge remote) \vee (couch) \vee (bench \wedge boat) \vee (traffic\_light \wedge giraffe) \vee (umbrella \wedge bowl)$
55. $(donut \wedge suitcase \wedge motorcycle) \vee (cup \wedge car \wedge orange) \vee (spoon) \vee (sink) \vee (horse) \vee (sandwich \wedge cell\_phone) \vee (couch)$
56. $(remote \wedge sports\_ball) \vee (skis \wedge kite \wedge vase) \vee (sheep \wedge cat \wedge bench) \vee (toothbrush) \vee (umbrella \wedge bottle \wedge suitcase) \vee (truck \wedge snowboard) \vee (toilet) \vee (bed)$
57. $(boat \wedge chair \wedge laptop) \vee (sports\_ball \wedge spoon \wedge kite) \vee (apple \wedge frisbee) \vee (baseball\_glove \wedge cow \wedge bowl) \vee (zebra \wedge dog \wedge truck) \vee (bench) \vee (skis)$
58. $(fire\_hydrant \wedge dog \wedge zebra) \vee (skateboard \wedge microwave \wedge book) \vee (toilet) \vee (motorcycle) \vee (banana \wedge vase) \vee (orange \wedge knife) \vee (cat \wedge bottle) \vee (wine\_glass \wedge cup)$
59. $(spoon \wedge cup \wedge cow) \vee (fire\_hydrant) \vee (teddy\_bear) \vee (giraffe) \vee (donut \wedge dog) \vee (airplane) \vee (surfboard \wedge chair) \vee (skis \wedge laptop \wedge bear) \vee (sports\_ball \wedge bicycle)$
60. $(bird) \vee (broccoli) \vee (bowl) \vee (sheep) \vee (couch \wedge elephant \wedge scissors) \vee (snowboard \wedge motorcycle) \vee (person \wedge dining\_table) \vee (surfboard \wedge baseball\_glove) \vee (teddy\_bear) \vee (mouse \wedge knife)$
61. $(suitcase \wedge knife) \vee (vase \wedge car) \vee (laptop \wedge giraffe) \vee (cell\_phone \wedge donut) \vee (airplane \wedge remote \wedge dining\_table) \vee (apple) \vee (spoon \wedge boat) \vee (tv \wedge sandwich \wedge carrot) \vee (stop\_sign \wedge orange) \vee (bed)$
62. $(scissors \wedge kite \wedge stop\_sign) \vee (oven \wedge cell\_phone \wedge wine\_glass) \vee (donut \wedge clock) \vee (zebra \wedge carrot \wedge skis) \vee (teddy\_bear \wedge sports\_ball \wedge frisbee) \vee (snowboard \wedge train) \vee (toothbrush \wedge laptop \wedge umbrella) \vee (apple)$
63. $(laptop) \vee (frisbee \wedge sink) \vee (pizza \wedge banana \wedge teddy\_bear) \vee (refrigerator \wedge toaster \wedge bench) \vee (boat) \vee (mouse \wedge tennis\_racket \wedge traffic\_light) \vee (sports\_ball \wedge knife) \vee (motorcycle) \vee (bus) \vee (baseball\_glove \wedge skateboard \wedge giraffe)$
64. $(sandwich) \vee (wine\_glass \wedge sink \wedge person) \vee (skateboard \wedge suitcase) \vee (dining\_table) \vee (toaster \wedge car) \vee (clock \wedge toothbrush) \vee (oven \wedge carrot) \vee (skis \wedge fork) \vee (bed) \vee (horse \wedge sports\_ball) \vee (pizza \wedge bear)$
65. $(frisbee) \vee (couch \wedge donut) \vee (fork \wedge microwave) \vee (tennis\_racket) \vee (oven \wedge backpack) \vee (toaster \wedge bear) \vee (train) \vee (orange \wedge spoon \wedge bench) \vee (dining\_table) \vee (bed) \vee (tv \wedge motorcycle \wedge horse) \vee (handbag)$
66. $(stop\_sign) \vee (person \wedge bear) \vee (chair) \vee (orange) \vee (sink \wedge cow) \vee (skateboard \wedge toilet) \vee (bowl \wedge frisbee \wedge skis) \vee (spoon \wedge fork) \vee (kite) \vee (bench \wedge truck) \vee (bird) \vee (donut \wedge laptop) \vee (tv) \vee (umbrella \wedge refrigerator \wedge toothbrush)$
67. $(bottle \wedge knife \wedge tennis\_racket) \vee (mouse \wedge baseball\_bat) \vee (toilet) \vee (couch) \vee (oven) \vee (skis) \vee (cat) \vee (refrigerator \wedge horse) \vee (cell\_phone \wedge clock \wedge cup) \vee (skateboard \wedge backpack \wedge handbag) \vee (cake) \vee (snowboard) \vee (stop\_sign \wedge bus \wedge laptop) \vee (bicycle)$

68. $(sink \wedge fork) \vee (suitcase \wedge toaster \wedge scissors) \vee (train) \vee (snowboard \wedge potted\_plant) \vee (teddy\_bear) \vee (toilet) \vee (wine\_glass \wedge parking\_meter \wedge vase) \vee (apple \wedge carrot) \vee (handbag \wedge sports\_ball \wedge toothbrush) \vee (sandwich \wedge surfboard) \vee (bed) \vee (person \wedge baseball\_bat) \vee (cup)$
69. $(mouse) \vee (vase \wedge carrot) \vee (bear \wedge tennis\_racket \wedge skis) \vee (clock \wedge spoon \wedge oven) \vee (banana \wedge scissors) \vee (sheep \wedge potted\_plant) \vee (pizza \wedge bicycle) \vee (surfboard \wedge broccoli) \vee (wine\_glass \wedge baseball\_bat) \vee (tie \wedge cat) \vee (teddy\_bear \wedge sports\_ball \wedge bottle)$
70. $(truck \wedge baseball\_bat) \vee (cat \wedge cup) \vee (stop\_sign \wedge parking\_meter \wedge surfboard) \vee (remote \wedge laptop \wedge elephant) \vee (tie \wedge bench \wedge car) \vee (sink) \vee (backpack \wedge bowl \wedge couch) \vee (dog) \vee (traffic\_light \wedge motorcycle) \vee (sports\_ball \wedge toilet) \vee (sheep \wedge bottle)$

B.3. 4 predicate query list

The following queries were used for the experiments in section 5.3:

1. $(zebra \vee handbag) \wedge (parking\_meter) \wedge (skis)$
2. $(sheep) \wedge (fork) \wedge (couch \vee horse)$
3. $(skateboard \vee dog \vee baseball\_glove) \wedge (skis)$
4. $(couch \vee skis) \wedge (fork \vee apple)$
5. $(spoon \vee toilet) \wedge (skateboard \vee backpack)$
6. $(zebra \wedge handbag) \vee (parking\_meter) \vee (skis)$
7. $(sheep) \vee (fork) \vee (couch \wedge horse)$
8. $(skateboard \wedge dog \wedge baseball\_glove) \vee (skis)$
9. $(couch \wedge skis) \vee (fork \wedge apple)$
10. $(spoon \wedge toilet) \vee (skateboard \wedge backpack)$

B.4. NLP query list

The following queries were used for the practical experiments in section 5.4:

1. $(positive \vee obscene)$
2. $(positive \wedge obscene)$
3. $(threat \vee negative)$
4. $(threat \wedge negative)$
5. $(insult) \wedge (severe\_toxic)$
6. $(insult) \vee (severe\_toxic)$
7. $(obscene) \wedge (toxic)$
8. $(obscene) \vee (toxic)$
9. $(neutral \vee toxic)$
10. $(neutral \wedge toxic)$
11. $(obscene) \wedge (toxic) \wedge (neutral) \wedge (identity\_hate)$
12. $(obscene) \vee (toxic) \vee (neutral) \vee (identity\_hate)$
13. $(threat \vee severe\_toxic \vee neutral) \wedge (obscene)$
14. $(threat \wedge severe\_toxic \wedge neutral) \vee (obscene)$
15. $(negative) \wedge (identity\_hate \vee toxic \vee insult)$
16. $(negative) \vee (identity\_hate \wedge toxic \wedge insult)$
17. $(negative) \wedge (insult \vee toxic) \wedge (threat)$
18. $(negative) \vee (insult \wedge toxic) \vee (threat)$
19. $(negative \vee threat) \wedge (insult) \wedge (neutral)$
20. $(negative \wedge threat) \vee (insult) \vee (neutral)$
21. $(obscene) \wedge (positive \vee negative) \wedge (severe\_toxic \vee toxic \vee identity\_hate)$
22. $(obscene) \vee (positive \wedge negative) \vee (severe\_toxic \wedge toxic \wedge identity\_hate)$
23. $(obscene \vee positive) \wedge (severe\_toxic \vee negative \vee toxic) \wedge (identity\_hate)$

24. $(obscene \wedge positive) \vee (severe\_toxic \wedge negative \wedge toxic) \vee (identity\_hate)$
25. $(identity\_hate \vee negative \vee threat) \wedge (obscene \vee toxic) \wedge (positive)$
26. $(identity\_hate \wedge negative \wedge threat) \vee (obscene \wedge toxic) \vee (positive)$
27. $(positive) \wedge (insult) \wedge (neutral \vee obscene) \wedge (identity\_hate) \wedge (toxic)$
28. $(positive) \vee (insult) \vee (neutral \wedge obscene) \vee (identity\_hate) \vee (toxic)$
29. $(negative \vee neutral) \wedge (insult \vee threat) \wedge (positive \vee obscene)$
30. $(negative \wedge neutral) \vee (insult \wedge threat) \vee (positive \wedge obscene)$
31. $(toxic \vee negative) \wedge (insult \vee neutral) \wedge (threat) \wedge (identity\_hate) \wedge (positive \vee severe\_toxic)$
32. $(toxic \wedge negative) \vee (insult \wedge neutral) \vee (threat) \vee (identity\_hate) \vee (positive \wedge severe\_toxic)$
33. $(negative \vee severe\_toxic) \wedge (insult \vee neutral) \wedge (toxic \vee identity\_hate \vee threat) \wedge (positive)$
34. $(negative \wedge severe\_toxic) \vee (insult \wedge neutral) \vee (toxic \wedge identity\_hate \wedge threat) \vee (positive)$
35. $(insult \vee obscene \vee negative) \wedge (neutral \vee threat) \wedge (identity\_hate) \wedge (severe\_toxic \vee toxic)$
36. $(insult \wedge obscene \wedge negative) \vee (neutral \wedge threat) \vee (identity\_hate) \vee (severe\_toxic \wedge toxic)$
37. $(toxic \vee insult \vee neutral) \wedge (obscene) \wedge (severe\_toxic) \wedge (negative \vee identity\_hate) \wedge (threat)$
38. $(toxic \wedge insult \wedge neutral) \vee (obscene) \vee (severe\_toxic) \vee (negative \wedge identity\_hate) \vee (threat)$
39. $(toxic) \wedge (threat \vee obscene) \wedge (neutral \vee positive) \wedge (severe\_toxic \vee insult \vee negative)$
40. $(toxic) \vee (threat \wedge obscene) \vee (neutral \wedge positive) \vee (severe\_toxic \wedge insult \wedge negative)$

C

Model zoo

From the datasets we derive the following (normalized) selectivity:

class	selectivity
toxic	0.078605
severe_toxic	0.008201
obscene	0.043455
threat	0.002459
insult	0.040482
identity_hate	0.00719
negative	0.337772
neutral	0.38271
positive	0.099126

model_index	cost	memory	*toxic*	*severe_toxic*	*obscene*	*threat*	*insult*	*identity_hate*	*negative*	*neutral*	*positive*
0	3	32652732	0.7573	0	0	0	0	0	0	0	0
1	3	32652732	0	0.35294	0	0	0	0	0	0	0
2	3	32652732	0	0	0.78745	0	0	0	0	0	0
3	3	32652732	0	0	0	0.34981	0	0	0	0	0
4	3	32652732	0	0	0	0	0.67763	0	0	0	0
5	3	32652732	0	0	0	0	0	0.3302	0	0	0
6	3	32652732	0	0	0	0	0	0	0.76241	0	0
7	3	32652732	0	0	0	0	0	0	0	0.75445	0
8	3	32652732	0	0	0	0	0	0	0	0	0.66288
9	47	38690347	0.75565	0.43806	0	0	0	0	0	0	0
10	46	38684015	0.75856	0	0.80251	0	0	0	0	0	0
11	47	38653451	0.75693	0	0	0.41846	0	0	0	0	0
12	49	38612867	0.76726	0	0	0	0.71142	0	0	0	0
13	46	38674619	0.75853	0	0	0	0	0.41657	0	0	0
14	47	38714790	0	0.38141	0.79279	0	0	0	0	0	0
15	51	38612526	0	0.394	0	0.34921	0	0	0	0	0
16	46	38654995	0	0.41857	0	0	0.69	0	0	0	0
17	47	38716411	0	0.40037	0	0	0	0.38402	0	0	0
18	50	38695603	0	0	0.79659	0.35556	0	0	0	0	0
19	47	38702214	0	0	0.79902	0	0.68206	0	0	0	0
20	50	38650434	0	0	0.78265	0	0	0.41083	0	0	0
21	48	38662665	0	0	0	0.42697	0.69664	0	0	0	0
22	50	38671389	0	0	0	0.44375	0	0.41215	0	0	0
23	47	38672973	0	0	0	0	0.68766	0.39629	0	0	0
24	50	38690347	0	0	0	0	0	0	0.78389	0.76423	0
25	49	38684015	0	0	0	0	0	0	0.79365	0	0.70293
26	50	38653451	0	0	0	0	0	0	0	0.76328	0.69452
27	48	38751127	0.76368	0.43082	0.65052	0.04865	0	0	0	0	0
28	47	38744534	0.76815	0.36308	0.65711	0	0.33874	0	0	0	0
29	48	38712254	0.75717	0.44573	0.65375	0	0	0.14271	0	0	0
30	48	38669186	0.7575	0.3979	0	0	0.32899	0	0	0	0
31	47	38734706	0.76511	0.42321	0	0.00423	0	0.15576	0	0	0
32	48	38776854	0.76285	0.4142	0	0	0.64338	0.1194	0	0	0
33	49	38668674	0.76757	0	0.80797	0.0065	0.68539	0	0	0	0
34	48	38713882	0.7604	0	0.81562	0	0	0.12591	0	0	0
35	48	38778646	0.76663	0	0.81252	0	0.62061	0.08577	0	0	0
36	50	38756722	0.7665	0	0	0	0.59642	0.19015	0	0	0
37	48	38763261	0	0.42445	0.79895	0.0383	0.65672	0	0	0	0
38	50	38708481	0	0.38246	0.78438	0.02989	0	0.11623	0	0	0
39	50	38721645	0	0.39925	0.80225	0	0.31808	0.09723	0	0	0
40	50	38730717	0	0.33367	0	0.2314	0.30503	0.18085	0	0	0
41	48	38732553	0	0	0.79434	0.09434	0.65038	0.12431	0	0	0

Table C.1: The model zoo generated for the NLP tasks