An Evaluation of Models for Runtime Approximation in Link Discovery

Kleanthi Georgala
AKSW Research Group, University of Leipzig
Augustusplatz 10
Leipzig, Germany 04109
georgala@informatik.uni-leipzig.de

Michael Hoffmann
AKSW Research Group, University of Leipzig
Augustusplatz 10
Leipzig, Germany 04109
mhoffmann@informatik.uni-leipzig.de

Axel-Cyrille Ngonga Ngomo
Paderborn University, Data Science Group
Pohlweg 51
Pacerborn, Germany D-33098
ngonga@upb.de

ABSTRACT

Time-efficient link discovery is of central importance to implement the vision of the Semantic Web. Some of the most rapid Link Discovery approaches rely internally on planning to execute link specifications. In newer works, linear models have been used to estimate the runtime of the fastest planners. However, no other category of models has been studied for this purpose so far. In this paper, we study non-linear runtime estimation functions for runtime estimation. In particular, we study exponential and mixed models for the estimation of the runtimes of planners. To this end, we evaluate three different models for runtime on six datasets using 500 link specifications. We show that exponential and mixed models achieve better fits when trained but are only to be preferred in some cases. Our evaluation also shows that the use of better runtime approximation models has a positive impact on the overall execution of link specifications.

CCS CONCEPTS

• **Information systems → Information integration**; **World Wide Web**;

KEYWORDS

Link Discovery, Taylor Series, Runtime Approximation, Link Specifications

ACM Reference format:
Kleanthi Georgala, Michael Hoffmann, and Axel-Cyrille Ngonga Ngomo. 2017. An Evaluation of Models for Runtime Approximation in Link Discovery. In *Proceedings of WI '17, Leipzig, Germany, August 23-26, 2017,* 8 pages.
DOI: 10.1145/3106426.3106428

1 INTRODUCTION

Link discovery frameworks are of utmost importance during the creation of Linked Data [1]. This is due to their being the key towards the implementation of the fourth Linked Data principle, i.e., the provision of links between datasets.[1] Two main challenges need to be addressed by Link Discovery frameworks [13, 14]. First, they need to address the *accuracy challenge*, i.e., they need to generate correct links. A plethora of approaches have been developed for this purpose and contain algorithms ranging from genetic programming to probabilistic models. In addition to addressing the need for accurate links, link discovery frameworks need to address the *challenge of time efficiency*. This challenge comes about because of the mere size of knowledge bases that need to be linked. In particular, large knowledge bases such as LinkedTCGA [17] contain more than 20 billion triples.

One of the approaches to improving the scalability of link discovery frameworks is to use planning algorithms in a manner akin (but not equivalent to) their use in databases [14]. In general, planners rely on cost functions to estimate the runtime of particular portions of link specifications. So far, it has been assumed that this cost function is linear in the parameters of the planning, i.e., in the size of the datasets and the similarity threshold. However, this assumption has never been verified. In this paper, we address exactly this research gap and *study how well other models for runtime approximation perform.* In particular, we study linear, exponential and mixed models for runtime estimation. The contributions of this paper are thus as follows: (1) We present three different models for runtime approximation in planning for Link Discovery. (2) We compare these models on six different datasets and study how well they can approximate runtimes of specifications as well as with respect to how well they generalize across datasets. (3) We integrate the models with the Helios planner for Link Discovery as described in [14] and compare their performance using 500 specifications.

The rest of the paper is structured as follows: In Section 2, we present the concept and notations necessary to understand this work. The subsequent section, Section 3, presents the runtime approximation problem and how it can be addressed by different models. We then delve into a thorough evaluation of these models in Section 4 and compare the expected runtimes generated by the models at hand with the real runtimes of the Link Discovery framework. We also study the transferability of the results we achieve and their performance when planning whole link specifications. Finally, we recapitulate our results and conclude.

[1]https://www.w3.org/DesignIssues/LinkedData.html

2 PRELIMINARIES

In this section, we present the necessary concepts and notations to understand the rest of the paper. We begin by giving a description of a knowledge base K and Link Discovery (LD), we continue by providing a formal definition of a link specification (LS) and its semantics and we finish our preliminary section with an explanatory presentation of a plan, its components and its relation to a LS.

Knowledge Base. A knowledge base K is a set of triples $(s, p, o) \in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{P} \times (\mathcal{R} \cup \mathcal{B} \cup \mathcal{L})$, where $\mathcal{R}$ is the set of all RDF resources, $\mathcal{P} \subseteq \mathcal{R}$ is the set of all RDF properties, $\mathcal{B}$ is the set of all RDF blank nodes and $\mathcal{L}$ is the set of all literals.

Link Discovery. Given two (not necessarily distinct) sets of RDF resources S and T and a relation R (e.g, `directorOf`, `owl:sameAs`), the main goal of LD is to discover the set (*mapping*) $\{(s, t) \in S \times T : R(s, t)\}$. Given that this task can be very tedious (especially when S and T are large), LD frameworks are commonly used to achieve this computation.

Link Specification. Declarative LD frameworks use link specifications (LSs) to describe the conditions for which $R(s, t)$ holds for a pair $(s, t) \in S \times T$. A LS consists of two basic components:

- *similarity measures* which allow the comparison of property values of resources found in the input datasets S and T. We define an *atomic similarity measure* $m \in M$ as a function $m : S \times T \times \mathcal{P}^2 \rightarrow [0, 1]$. We write $m(s, t, p_s, p_t)$ to signify the similarity of s and t w.r.t. their properties p_s resp. p_t.
- *operators* $op \in \{\sqcup, \sqcap, \backslash\}$ that allow the combination of two *similarity measures.*

An atomic LS consists of one similarity measure and has the form $(m(p_s, p_t), \theta)$ where $\theta \in [0, 1]$. A complex LS $L = op(L_1, L_2)$ consists of two LS, L_1 and L_2. We call L_1 the *left sub-specification* and L_2 the *right sub-specification* of L. We denote the semantics (i.e., the results of a LS for given sets of resources S and T) of a LS L as $[[L]]$ and call it a mapping. We begin by assuming the natural semantics of the combinations of measures. Filters are pairs (f, τ), where (1) f is either empty (denoted ϵ) or a combination of similarity measures by means of specification operators and (2) τ is a threshold. Note that an atomic specification can be regarded as a filter (f, τ, X) with $[[X]] = S \times T$. We will thus use the same graphical representation for filters and atomic specifications. We call (f, τ) the *filter of L* and denote it with $\varphi(L)$. For our example L in Fig. 1, $\varphi(L) = (\epsilon, 0.7)$. We denote the *operator of a LS L* with $op(L)$. For $L = (f, \tau, \omega(L_1, L_2))$, $op(L) = \omega$. The operator of the LS shown in our example is $\sqcup$. The semantics of LSs are then as shown in Table 1.

Execution Plan. To compute the mapping $[[L]]$ (which corresponds to the output of L for a given pair (S, T)), LD frameworks implement (at least partly) a generic architecture consisting of a rewriter (optional), a planner (optional) and an execution engine (necessary). The *rewriter* performs algebraic operations to transform the input LS L into a LS L' (with $[[L]] = [[L]]'$) that is potentially faster to execute. The most common planner is the *canonical planner* (dubbed CANONICAL), which simply traverses L in post-order and has its results computed in that order by the execution engine.[2] For the LS shown in Fig. 1, the execution plan returned by CANONICAL would thus foresee to first compute the mapping $M_1 = [[(\texttt{trigrams}(\texttt{:title}, \texttt{:title}), 0.48)]]$ of pairs of resources whose property `title` has a cosine similarity greater or equal to 0.48. The computation of $M_2 = [[(\texttt{levenSim}(\texttt{:label}, \texttt{:label}), 0.46)]]$ would follow. Step 3 would be to compute $M_3 = M_1 \sqcap M_2$ while abiding by the semantics described in Table 1. Step 4 would be to obtain M_4 by filtering the results and keeping only the pairs that have a similarity above 0.5. Step 5 would be $M_5 = [[(\texttt{cosine}(\texttt{:name}, \texttt{:name}), 0.78)]]$ and Step 6 would be to compute $M_6 = M_4 \sqcup M_5$. Finally, Step 7 would be to filter out the pairs of links in M_6 that have a similarity less than 0.8. Given that there is a 1-1 correspondence between LS and the plan generated by the canonical planner, we will reuse the representation of LS devised above for plans. The sequence of steps for such a plan is then to be understood as the sequence of steps that would be derived by CANONICAL for the LS displayed.

Table 1: Semantics of link specifications

L	$[[L]]$
(m, θ)	$\{(s, t, m(s, t)) \in S \times T : m(s, t) \geq \theta\}$
(f, τ, X)	$\begin{cases} \{(s, t, r) \in [[X]] : r \geq \tau\} \text{ if } f = \epsilon \\ \{(s, t, r) \in [[X]] : f(s, t) \geq \tau\} \text{ else.} \end{cases}$
$\sqcap(L_1, L_2)$	$\{(s, t, r) \mid (s, t, r_1) \in [[L_1]] \land (s, t, r_2) \in [[L_2]] \land r = \min(r_1, r_2)\}$
$\sqcup(L_1, L_2)$	$\left\{ (s, t, r) \mid \begin{cases} r = r_1 \text{ if } \exists (s, t, r_1) \in [[L_1]] \land \neg(\exists r_2 : (s, t, r_2) \in [[L_2]]), \\ r = r_2 \text{ if } \exists (s, t, r_2) \in [[L_2]] \land \neg(\exists r_1 : (s, t, r_1) \in [[L_1]]), \\ r = \max(r_1, r_2) \text{ if } (s, t, r_1) \in [[L_1]] \land (s, t, r_2) \in [[L_2]]. \end{cases} \right\}$
$\backslash(L_1, L_2)$	$\{(s, t, r) \mid (s, t, r) \in [[L_1]] \land \neg \exists r' : (s, t, r') \in [[L_2]]\}$
$\emptyset(L)$	$[[L]]$

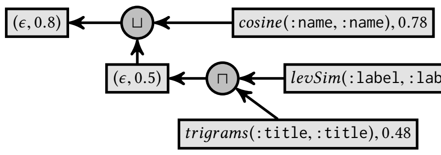
(ϵ, 0.8)
⊔
cosine(:name, :name), 0.78
(ϵ, 0.5)
⊓
levSim(:label, :label), 0.46
trigrams(:title, :title), 0.48

Figure 1: Graphical representation of an example LS

3 RUNTIME ESTIMATION

In general, planners aims to estimate the cost of the leaves of a plan, i.e., the runtime of atomic link specifications. So far, linear models [14] have been used for this purpose but the appropriateness of other models has never been evaluated. Hence, in this work, we compare non-linear models with linear models to approximate the runtime of of atomic link specifications. Like in previous works, we follow a *sampling-based approach.* First, given a particular similarity measure m (e.g., Levenshtein) and an implementation of the said measure (e.g., *Ed-Join* [21]), we begin by collecting sample of runtimes for a given measure with varying values of $|S|$, $|T|$ and θ.[3] These samples can be regarded as the output of a function that can predict the runtime of the implementation of m for which we

[2]Note that the planner and engine are not necessarily distinct in existing implementations.
[3]We also experimented with the number of trigrams contained in S and T but found that they do not affect the models we considered. An exploration of other parameters remains future work.

were given samples. The major question that is to be answered is hence *what is the shape of the runtime evaluation function*?

We tried fitting functions of different shapes to the previously measured runtimes in order to compare their performance when planning the execution of link specifications. Formally, these functions are mappings $\phi : \mathbb{N}\times\mathbb{N}\times(0,1] \mapsto \mathbb{R}$, whose value at $(|S|, |T|, \theta)$ is an approximation of the runtime for the link specification with these parameters. If $\vec{R} = (R_1, \dots, R_n)$ are the measured runtimes for the parameters $\vec{S} = (|S_1|, \dots, |S_n|)$, $\vec{T} = (|T_1|, \dots, |T_n|)$ and $\vec{\theta} = (\theta_1, \dots, \theta_n)$, then we constrain the mapping ϕ to be a local minimum of the L2-Loss:

$$E(\vec{S}, \vec{T}, \vec{\theta}, \vec{r}) := ||\vec{R} - \phi(\vec{S}, \vec{T}, \vec{\theta})||^2, \quad (1)$$

writing $\phi(\vec{S}, \vec{T}, \vec{\theta}) = (\phi(|S_1|, |T_1|, \theta_1), \dots, \phi(|S_n|, |T_n|, \theta_n))$.

Within this paper, we consider the following parametrized families of functions:

$$\begin{aligned}
\phi_1(S, T, \theta) &= a + b|S| + c|T| + d\theta && (2)\\
\phi_2(S, T, \theta) &= \exp\left(a + b|S| + c|T| + d\theta + e\theta^2\right) && (3)\\
\phi_3(S, T, \theta) &= a + (b + c|S| + d|T| + e|S||T|)\exp\left(f\theta + g\theta^2\right) && (4)
\end{aligned}$$

The parameters are then determined by

$$a^*, b^*, \dots = \arg\min E(\vec{S}, \vec{T}, \vec{\theta}, \vec{R})(a, b, \dots) \quad (5)$$

for some local minimum. In the case of ϕ_1 and ϕ_2 this problem is linear in nature and we solved it using the pseudo-inverse of the associated Vandermonde matrix. For ϕ_3 we used the Levenberg-Marquardt Algorithm [11] for nonlinear least squares problems, using 1 as initial guess for all parameters.

We chose ϕ_1 as the baseline linear fit. ϕ_2 is the standard log-linear fit, except for the θ^2 term. We included this term during a grid search for polynomials to perform a log-polynomial fit. Higher orders of $|S|$ or $|T|$ or θ did not contribute to a better fit. ϕ_3 can be interpreted as an interpolation of ϕ_1 and ϕ_2 with a constant offset a.

To exemplify our approach for ϕ_2, assume we have measured $\vec{S} = (458, 458, 358, 58)$, $\vec{T} = (512, 404, 317, 512)$ and $\vec{\theta} = (0.5, 0.9, 0.6, 0.7)$. Inserting into eq. (1) and taking the logarithm, one arrives at the optimization problem

$$\min_{a,b,c,d,e} \left|\left| \begin{pmatrix} 1 & 458 & 512 & 0.5 & 0.5^2 \\ 1 & 458 & 404 & 0.9 & 0.9^2 \\ 1 & 358 & 317 & 0.6 & 0.6^2 \\ 1 & 58 & 512 & 0.7 & 0.7^2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} - \begin{pmatrix} \log(67) \\ \log(4) \\ \log(4) \\ \log(1) \end{pmatrix} \right|\right|^2$$

The solution to this least squares problem also is the unique solution of its normal equations:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 458 & 458 & 358 & 58 \\ 512 & 404 & 317 & 512 \\ 0.5 & 0.9 & 0.6 & 0.7 \\ 0.5^2 & 0.9^2 & 0.6^2 & 0.7^2 \end{pmatrix} \begin{pmatrix} 1 & 458 & 512 & 0.5 & 0.5^2 \\ 1 & 458 & 404 & 0.9 & 0.9^2 \\ 1 & 358 & 317 & 0.6 & 0.6^2 \\ 1 & 58 & 512 & 0.7 & 0.7^2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 458 & 458 & 358 & 58 \\ 512 & 404 & 317 & 512 \\ 0.5 & 0.9 & 0.6 & 0.7 \\ 0.5^2 & 0.9^2 & 0.6^2 & 0.7^2 \end{pmatrix} \begin{pmatrix} \log(67) \\ \log(4) \\ \log(4) \\ \log(1) \end{pmatrix}$$

By multiplying and inverting matrices, we arrive at the linear equation

$$\begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} 1 & 458 & 512 & 0.5 & 0.5^2 \\ 1 & 458 & 404 & 0.9 & 0.9^2 \\ 1 & 358 & 317 & 0.6 & 0.6^2 \\ 1 & 58 & 512 & 0.7 & 0.7^2 \end{pmatrix}^+ \begin{pmatrix} \log(67) \\ \log(4) \\ \log(4) \\ 0 \end{pmatrix},$$

where A^+ denotes the Moore-Penrose pseudo inverse of A [5]. Multiplying the matrices, we arrive at

$$\begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} -1.028 \\ 0.009 \\ 0.010 \\ 9.821 \\ -9.053 \end{pmatrix}.$$

Thus we have found the coefficients of the fit function.

4 EVALUATION

4.1 Experimental Setup

We evaluated the three runtime estimation models using six datasets. The first three are the benchmark datasets for LD dubbed Amazon-Google Products, DBLP-ACM and DBLP-Scholar described in [10]. We also created two larger additional datasets (MOVIES and VILLAGES, see Table 2) from the datasets DBpedia, LinkedGeodata and LinkedMDB. [4] [5] The sixth dataset was the set of all English labels from DBpedia 2014. Table 2 describes the characteristics of the datasets and presents the properties used when linking the retrieved resources for the first four datasets. The mapping properties were provided to the link discovery algorithms underlying our results.

Each of our experiments consisted of two phases: During the *training* phase, we trained each of the models independently. For each model, we computed the set of coefficients for each of the approximation models that minimized the root mean squared error (RMSE) on the training data provided. The aim of the subsequent *test* phase was to evaluate the accuracy of the runtime estimation provided by each model and the performance of the currently best LD planner, HELIOS [14], when it relied of each of the three models for runtime approximations. Throughout our experiments, we used the algorithms *Ed-Join* [22] (which implements the Levenshtein string distance) and *PPJoin+* [23] (which implements the Jaccard, Overlap, Cosine and Trigrams string similarity measures) to execute atomics specifications. As thresholds θ we used random values between 0.5 and 1.

The aim of our evaluation was to answer the following set of questions regarding the performance of the three models *exp*, *linear* and *mixed*: [6]

- Q_1: How do our models fit each class separately? To answer this question, we began by splitting the source and target data of each of our datasets into two non-overlapping parts of equal size. We used the first half of each source and each target for training and the second half for testing. *Training*:

[4]http://www.linkedmdb.org/

[5]The new datasets as well as a description of how they were constructed are available at http://titan.informatik.uni-leipzig.de/kgeorgala/DATA/.

[6]For Q_1 and Q_2 we did not conduct experiments using the dataset derived from DBPedia english labels, since it includes labels from multiple classes.

We trained the three models on each dataset. For each model, dataset and mapper, we a) selected 15 source and 15 target random samples of random sizes from the first half of a dataset (Amazon-Google Products, DBLP-ACM, DBLP-Scholar, MOVIES and VILLAGES) and b) compared each source sample with each target sample 3 times. Note that we used the same samples across all models for the sake of fairness. Overall, we ran 675 training experiments to train each model on each dataset. *Testing*: To test the accuracy of each model, we ran the corresponding algorithm (*Ed-Join* and *PPJoin+*) with a random threshold between 0.5 and 1 and recorded the real runtime of the approach and the runtimes predicted by our three models. Each approach was executed 100 times against the whole of the second half of the same dataset.

- Q_2: How do our models generalize across classes, i.e., can a model trained on data from one class be used to predict runtimes accurately on another class? *Training*: We trained each model in the same manner as for Q_1 on exactly the same five datasets with the sole difference that the samples were selected randomly from the whole dataset. *Testing*: Like in the previous series of experiments, we ran *Ed-Join* and *PPJoin+* with a random threshold between 0.5 and 1. Each of the algorithms was executed 100 times against the remaining four datasets.
- Q_3: How do our models perform when trained on a large dataset? *Training*: We trained in the same fashion as to answer Q_1 with the sole differences that (1) we used 15 source and 15 target random samples of various sizes between $10,000$ and $100,000$ from (2) the English labels of DBpedia to train our model. *Testing*: We learned 100 LSs for the Amazon-GP, DBLP-ACM, MOVIES and VILLAGES datasets using the unsupervised version of the EAGLE algorithm [12]. We chose this algorithm because it was shown to generate meaningful specifications that return high-quality links in previous works. For each dataset, we ran the set of 100 specifications learned by EAGLE on the given dataset by using each of the models during the execution in combination with the HELIOS planning algorithm [14], which was shown to outperforms the canonical planner w.r.t. runtime while producing exactly the same results.

Throughout our experiments, we configured EAGLE by setting the number of generations and population size to 20, mutation and crossover rates were set to 0.6. All experiments for all implementations were carried out on the same 20-core Linux Server running *OpenJDK* 64-Bit Server 1.8.0_74 on Ubuntu 14.04.4 LTS on Intel(R) Xeon(R) CPU E5-2650 v3 processors clocked at 2.30GHz. Each *train* experiment and each *test* experiment for Q_3 was repeated three times. As evaluation measure, we computed root mean square error (*RMSE*) between the *expected* runtime and the average *execution* runtime required to run each LS. We report all three numbers for each model and dataset.

4.2 Results

To address Q_1, we evaluated the performance of our models when trained and tested on the same class. We present the results of this series of experiments in Table 3. For *PPJoin+* (in particular the *trigrams* measure), the *mixed* model achieved the lowest error when tested upon Amazon-GP and DBLP-Scholar, whereas the *linear* model was able to approximate the expected runtime with higher accuracy on the MOVIES and VILLAGES datasets. On average, *linear* model was able to achieve a lower *RMSE* compared to the other two models. For the *Ed-Join*, the *mixed* model outperformed *linear* and *exp* in the majority of datasets (DBLP-Scholar, MOVIES and VILLAGES) and obtained the lowest *RMSE* on average. As we observe in Table 3, for both measures, the *exp* model retrieved the highest error on average and is thus the model less suitable for runtime approximations. Especially, for the *Ed-Join*, *exp* had the worst performance in four out of the five datasets and retrieved the highest *RMSE* among the different test datasets for VILLAGES. This clearly answers our first questions: the *linear* and *mixed* approximation models are able achieve the smallest error when trained on the class on which they are tested.

To continue with Q_2, we conducted a set of experiments in order to observe how well each model could generalize among the different classes included in our evaluation data. Tables 4, 5, 6, 7 and 8 present the results of training on one dataset and testing the resulting models on the set of the remaining classes. tab:exp2AmazonThe highest *RMSE* error was achieved when both measures were tested using the *exp* model in all datasets but VILLAGES. However, Table 8 shows that the fitting error when trained on VILLAGES is relatively low among all three models. Additionally, we observe that the *exp* model's *RMSE* increased exponentially as the quantity of the training data decreased, which constitutes this model as inadequate and unreliable for runtime approximations. By observing Tables 5 and 6, we see that the *RMSE* of the *exp* model increased by 38 orders of magnitude for *Ed-Join*.

For both measures, the *linear* model outperformed the other two models on average when trained on the Amazon-GP, DBLP-ACM and DBLP-Scholar datasets and achieved the lowest *RMSE* when trained on MOVIES for *Ed-Join* compared to *exp* and *mixed*. Both *linear* and *mixed* achieved minuscule approximation errors compared to *exp*, but *linear* was able to produce at least 35% less *RMSE* compared to *mixed*. Therefore, we can answer Q_2 by stating that the *linear* model is the most suitable and sufficient model that can generalize among different classes.

For our last question, we tested the performance of the different models when trained on a bigger and more diverse dataset. Table 9 shows the results of our evaluation, where each model was trained on DBpedia english labels and tested on the the four evaluation datasets. The *linear* model error was 1 order of magnitude less than the *RMSE* obtained by *exp* and 3 orders of magnitude less compared to the *mixed* error. In all four datasets, the *mixed* model produced the highest *RMSE*. For the VILLAGES dataset, the *mixed* model's error was 1916 and 214 times higher compared to *linear* and *exp* resp. Figs. 2 and 3 present the plans produces by HELIOS for the LS MINUS(AND(levenshtein(x.descr,y.descr)|0.5045, trigrams(x.title, y.name) |0.4871)|0.2925), OR(levenshtein(x.descr,y.descr)|0.5045, trigrams(x.title, y.name) |0.4871)|0.2925) >=0.2925 of the Amazon-GP dataset, if the planner used the *exp* model and the *linear* or the *mixed* model resp. For the child LS AND(levenshtein(x.descr,y.descr)|0.5045, trigrams(x.title, y.name)|0.4871)|0.2925, the *linear* and the *mixed*

Table 2: Entity matching characteristics of datasets

Dataset	Source (S)	Target (T)	$\|S\| \times \|T\|$	Source Property	Target Property
Amazon-GP	Amazon	Google Products	4.40×10^6	product name, description manufacturer, price	product name, description manufacturer, price
DBLP-ACM	ACM	DBLP venue	6.00×10^6	title, authors year, venue	title authors year
DBLP-Scholar	DBLP	Google Scholar	0.17×10^9	title, authors venue, year	title, authors venue, year
MOVIES	DBpedia	LinkedMDB	0.17×10^9	dbp:name dbo:director/dbp:name dbo:producer/dbp:name dbp:writer/dbp:name rdfs:label	dc2:title movie:director/movie:director_name movie:producer/movie:producer_name movie:writer/movie:writer_name rdfs:label
VILLAGES	DBpedia	LGD	6.88×10^9	rdfs:label dbo:populationTotal geo:geometry	rdfs:label lgdo:population geom:geometry/agc:asWKT

Table 3: Average expected runtime, average execution time and root mean square error for the first five datasets for training and testing on the same class. All runtimes are presented in milliseconds.

Measures	Model	Amazon-GP			DBLP-ACM			DBLP-Scholar		
		expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**	**expected**	**execution**	**RMSE**
PPJoin+	**exp**	7.33	14.45	2.78	8.36	14.56	2.43	177.02	124.88	8.02
	linear	8.37	16.24	3.28	7.45	15.81	2.97	222.55	147.33	9.48
	mixed	6.09	13.45	2.70	6.12	16.83	3.56	129.63	149.82	6.69
Ed-Join	**exp**	22.81	27.33	3.89	34.33	36.84	3.49	428.93	324.79	12.31
	linear	17.99	26.04	2.60	25.29	35.85	3.35	354.97	404.06	9.65
	mixed	18.34	26.45	2.78	27.68	41.20	3.54	338.55	339.31	7.30
Measures	**Model**	**MOVIES**			**VILLAGES**			**AVERAGE**		
		expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**			
PPJoin+	**exp**	134.90	146.39	5.44	211.89	135.53	9.36	*PPJoin+*	**exp**	5.61
	linear	38.60	33.10	2.95	158.89	131.64	5.23		**linear**	**4.78**
	mixed	48.45	49.89	3.17	214.15	201.17	8.13		**mixed**	4.85
Ed-Join	**exp**	59.57	45.47	3.76	1,225.57	1,556.04	35.23	*Ed-Join*	**exp**	11.74
	linear	43.02	44.46	3.52	509.71	294.35	22.53		**linear**	8.33
	mixed	45.55	43.26	2.88	377.02	286.91	10.89		**mixed**	**5.48**

Table 4: Average expected runtime, average execution time and root mean square error for training on Amazon-GP dataset and testing on DBLP-ACM, DBLP-Scholar, MOVIES and VILLAGES. All runtimes are presented in milliseconds.

Measures	Model	DBLP-ACM			DBLP-Scholar			AVERAGE		
		expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**			
PPJoin+	**exp**	18.24	64.02	8.61	1.84E+17	1,609.71	1.84E+16	*PPJoin+*	**exp**	8.42E+35
	linear	25.42	87.68	12.23	409.98	474.82	20.59		**linear**	**24.68**
	mixed	44.67	137.54	18.72	270.33	339.06	20.02		**mixed**	90.07
Ed-Join	**exp**	62.62	142.76	15.67	5.34E+19	834.11	5.34E+18			
	linear	37.19	131.68	19.26	663.07	837.88	27.30			
	mixed	38.36	140.25	16.87	770.51	861.72	21.91			
Measures	**Model**	**MOVIES**			**VILLAGES**					
		expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**			
PPJoin+	**exp**	8.79E+05	95.28	8.79E+04	3.37E+37	352.77	3.37E+36	*Ed-Join*	**exp**	8.43E+41
	linear	133.06	202.34	11.32	853.58	331.61	54.62		**linear**	**28.01**
	mixed	136.17	98.58	6.37	3,507.19	360.03	315.15		**mixed**	54.49
Ed-Join	**exp**	1.26E+07	143.93	1.26E+06	9.75E+42	6,108.37	9.75E+41			
	linear	209.13	142.45	9.14	1,379.12	864.31	56.32			
	mixed	332.13	145.46	19.83	7,258.82	5,973.70	159.37			

model chose to execute only `trigrams(x.title, y.name)|0.4871)`

and use the other child as a filter. Moreover, the plan retrieved by

Table 5: Average expected runtime, average execution time and root mean square error for training on DBLP-ACM dataset and testing on Amazon-GP, DBLP-Scholar, MOVIES and VILLAGES. All runtimes are presented in milliseconds.

Measures	Model	Amazon-GP			DBLP-Scholar			AVERAGE		
		expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**			
	exp	21.51	61.69	9.93	1.29E+16	3,741.58	1.29E+15			
PPJoin+	**linear**	15.73	46.13	8.95	346.71	3,674.06	341.87		**exp**	3.99E+15
	mixed	44.09	120.62	12.82	534.41	1,833.07	139.71	*PPJoin+*	**linear**	**101.82**
	exp	85.53	92.78	8.02	2.82E+18	888.50	2.82E+17		**mixed**	531.95
Ed-Join	**linear**	56.95	90.10	7.91	950.61	883.01	25.97			
	mixed	58.29	96.63	8.48	1,472.52	881.22	63.72			
Measures	**Model**	**MOVIES**			**VILLAGES**					
		expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**			
	exp	8.05E+05	108.16	8.05E+04	1.47E+37	356.93	1.47E+36			
PPJoin+	**linear**	127.07	132.62	7.64	819.98	368.86	48.82		**exp**	9.3E+42
	mixed	159.36	120.74	8.92	2.14E+04	1,783.72	1,966.38	*Ed-Join*	**linear**	**53.95**
	exp	3.58E+07	156.97	3.58E+06	3.72E+44	6,329.54	3.72E+43		**mixed**	1,105.15
Ed-Join	**linear**	373.99	156.72	23.23	2,440.64	870.15	158.72			
	mixed	1,246.20	155.42	109.39	4.87E+04	6,411.76	4,239.01			

Table 6: Average expected runtime, average execution time and root mean square error for training on DBLP-Scholar dataset and testing on Amazon-GP, DBLP-ACM, MOVIES and VILLAGES. All runtimes are presented in milliseconds.

Measures	Model	Amazon-GP			DBLP-ACM			AVERAGE		
		expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**			
	exp	79.32	65.28	8.03	47.42	69.70	8.74			
PPJoin+	**linear**	-364.95	38.47	40.61	173.40	88.48	15.39		**exp**	4.56E+04
	mixed	-41.05	50.27	11.00	-148.99	88.14	26.03	*PPJoin+*	**linear**	**85.07**
	exp	113.56	80.90	8.67	113.43	139.78	16.74		**mixed**	427.54
Ed-Join	**linear**	44.49	79.97	10.67	37.70	144.33	22.36			
	mixed	40.13	73.76	8.98	40.94	141.33	18.84			
Measures	**Model**	**MOVIES**			**VILLAGES**					
		expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**			
	exp	110.41	94.69	6.31	1.82E+06	1,546.07	1.82E+05			
PPJoin+	**linear**	394.74	104.19	29.99	3,158.25	621.84	254.30		**exp**	1.10E+04
	mixed	66.96	85.61	6.76	1.82E+04	1,591.24	1,666.38	*Ed-Join*	**linear**	**54.57**
	exp	341.02	128.33	22.66	4.46E+05	6,069.92	4.41E+04		**mixed**	82.52
Ed-Join	**linear**	360.47	127.76	24.51	2,418.34	818.14	160.73			
	mixed	280.77	125.19	16.86	3,670.31	820.85	285.43			

Table 7: Average expected runtime, average execution time and root mean square error for training on MOVIES dataset and testing on Amazon-GP, DBLP-ACM, DBLP-Scholar and VILLAGES. All runtimes are presented in milliseconds.

Measures	Model	Amazon-GP			DBLP-ACM			AVERAGE		
		expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**			
	exp	19.53	71.55	7.89	46.89	127.70	15.90			
PPJoin+	**linear**	-45.99	42.58	10.51	57.73	120.70	23.93		**exp**	8.42E+06
	mixed	16.97	39.64	5.84	17.43	66.84	9.77	*PPJoin+*	**linear**	51.34
	exp	15.57	80.95	9.37	16.24	135.66	17.93		**mixed**	**37.99**
Ed-Join	**linear**	1.71	84.53	10.82	3.56	138.18	19.89			
	mixed	4.33	85.70	10.95	6.99	140.99	19.65			
Measures	**Model**	**DBLP-Scholar**			**VILLAGES**					
		expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**			
	exp	3,636.56	318.89	332.11	3.37E+08	634.17	3.37E+07			
PPJoin+	**linear**	372.82	1,315.61	102.21	1,064.96	389.93	68.69		**exp**	1.46E+06
	mixed	75.49	702.11	67.82	989.17	311.60	68.54	*Ed-Join*	**linear**	**25.91**
	exp	4,060.80	811.77	325.48	5.85E+07	767.66	5.85E+06		**mixed**	42.85
Ed-Join	**linear**	259.61	805.29	57.92	696.29	753.35	15.04			
	mixed	178.93	796.16	65.09	1,522.63	777 .00	75.74			

Table 8: Average expected runtime, average execution time and root mean square error for training on VILLAGES dataset and testing on Amazon-GP, DBLP-ACM, DBLP-Scholar and MOVIES. All runtimes are presented in milliseconds.

Measures	Model	Amazon-GP			DBLP-ACM			AVERAGE		
		expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**			
PPJoin+	**exp**	93.41	67.44	5.08	35.07	62.53	8.36	*PPJoin+*	**exp**	**10.16**
	linear	-192.27	24.57	21.87	-133.03	61.10	21.09		**linear**	22.91
	mixed	16.37	32.66	3.40	41.57	61.83	9.20		**mixed**	30.59
Ed-Join	**exp**	68.00	53.36	4.50	326.05	143.84	26.53			
	linear	-123.44	55.03	18.20	-677.4	133.63	82.36			
	mixed	231.61	50.46	18.51	136.49	139.30	15.95			
Measures	**Model**	**DBLP-Scholar**			**MOVIES**					
		expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**			
PPJoin+	**exp**	92.10	272.40	21.78	56.74	82.92	5.43	*Ed-Join*	**exp**	**21.59**
	linear	-39.98	277.56	34.10	-54.33	84.08	14.57		**linear**	54.56
	mixed	84.22	451.80	40.04	-26.91	651.50	69.71		**mixed**	32.75
Ed-Join	**exp**	316.66	784.7	49.85	138.63	114.50	5.46			
	linear	159.75	753.00	61.23	-438.84	122.89	56.44			
	mixed	1,737.75	945.09	81.94	255.96	116.42	14.61			

using the *exp* model for runtime approximations aims to execute both children LSs, which results into an overhead in the execution of the LS. It is obvious that the *linear* model achieved by far the lowest *RMSE* on average compared to the other two models, which concludes the answer to Q_3.

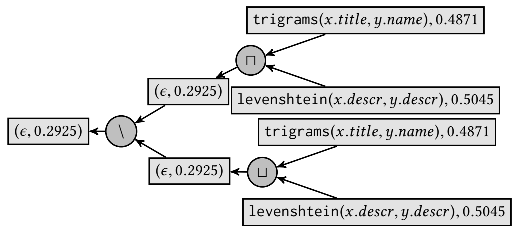
trigrams(x.title, y.name), 0.4871
⊓
(ε, 0.2925)
levenshtein(x.descr, y.descr), 0.5045
(ε, 0.2925)
\
trigrams(x.title, y.name), 0.4871
(ε, 0.2925)
⊔
levenshtein(x.descr, y.descr), 0.5045

Figure 2: Plan returned from Helios using the *exp* model.

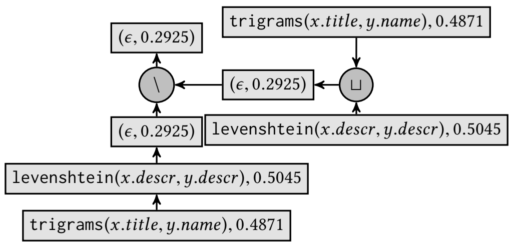
(ε, 0.2925)
trigrams(x.title, y.name), 0.4871
\
(ε, 0.2925)
⊔
(ε, 0.2925)
levenshtein(x.descr, y.descr), 0.5045
levenshtein(x.descr, y.descr), 0.5045
trigrams(x.title, y.name), 0.4871

Figure 3: Plan returned from Helios using the *linear* and *mixed* model.

5 RELATED WORK

The task of efficient query execution in database systems is similar to the task of execution optimization using runtime approximations in LD frameworks. Efficient and scalable data management has been of central importance in database systems [6]. Over the past few years, there has been an extensive work on query optimization in databases that is based on statistical information about relations and intermediate results [18]. The author of [3] gives an analytic overview regarding the procedure of query optimization and the different approaches used at each step of the process.

A novel approach in this field was presented by [7], in which the proposed approach introduced the concept of parametric query optimization. In this work, the authors provided the necessary formalization of the aforementioned concept and conducted a set of experiments using the buffer size as parameter. In order to minimize the total cost of generating all possible alternative execution plans, they used a set of randomized algorithms. On a similar manner, the authors of [19] introduced the idea of Multi-Objective Parametric query optimization (MPQ), where the cost of plan is associated with multiple cost functions and each cost function is associated with various parameters. Their experimental results showed however that the MPQ method performs an exhaustive search of the solution space which addresses this approach computationally inefficient.

Another set of approaches in the field of query optimization have focused on creating dynamic execution plans. Dynamic planning is based on the idea that the execution engine of a framework knows more than the planner itself. Therefore, information generated by the execution engine is used to re-evaluate the plans generated by the optimizer. There has been a vast amount of approaches towards dynamic query optimization such as query scrambling for initial delays [20], dynamic planning in compile-time [4], adaptive query operators [9] and re-ordering of operators [2].

Moreover, the problem addressed in this work focus on identifying scalable and time-efficient solutions towards LD. A large number of frameworks were developed to assist this issue, such as SILK [8], Limes [13], KnoFuss [15] and Zhishi.links [16]. SILK and KnoFuss implement blocking approaches in order to achieve efficient linking between resources. SILK framework incorporates a rough index pre-match, whereas KnoFuss blocking technique is highly influenced from databases systems techniques. To this end, the only LD framework that provides both theoretical and practical guarantees towards scalable and accurate LD is Limes. As we mentioned throughout this work,Limes execution strategy incorporates

Table 9: Average expected runtime, average execution time and root mean square error for training on DBPedia english labels and testing on Amazon-GP, DBLP-ACM, MOVIES and VILLAGES. All runtimes are presented in milliseconds.

Model	**Amazon-GP**				**DBLP-ACM**		**AVERAGE**	
	expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**		
exp	5,242.09	3,618.99	3,164.86	308.14	365.46	126.42		
linear	300.51	3,043.97	966.99	8.07	361.53	192.12		
mixed	-7.27E+06	4,512.82	6.78E+05	-7.26E+04	310.49	4.38E+04	**exp**	4,577.58
Model			**Amazon-GP**			**DBLP-ACM**	**linear**	**512.35**
expected	**execution**	**RMSE**	**expected**	**execution**	**RMSE**		**mixed**	9.82E+05
exp	584.27	1,061.67	160.05	4.61E+04	3,775.54	1.48E+04		
linear	323.04	995.04	258.55	2,626.41	3,832.52	631.72		
mixed	-3,417.80	1,600.81	2,042.45	7.15E+06	3,891.05	3.20E+06		

the Helios planner [14] which is (to the best of our knowledge) the first execution optimizer in LD. Helios is able to provide accurate runtime approximations, which we have extended in this work, and is able to find the least costly execution plan for a LS, consuming a minute portion of the overall execution runtime.

6 CONCLUSION

In this paper, we studied approximation functions that allow predicting the runtime of link specifications. We showed that on average, linear models are indeed the approach to chose to this end as they seem to overfit the least. Still, mixed models also perform in a satisfactory manner. Exponential models either fit very well or not at all and are thus not to be used. In future work, we will study further models for the evaluation of runtime and improve upon existing planning mechanisms for the declarative LD. In particular, we will consider other features when approximation runtimes, e.g., the distribution of characters in the strings to compare.

7 ACKNOWLEDGMENTS

This work has been supported by the H2020 project HOBBIT (GA no. 688227), the EuroStars project QAMEL (project no. 01QE1549C) and the BMWi project SAKE (project no. 01MD15006E).

REFERENCES

[1] Sören Auer, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, and Amrapali Zaveri. 2013. Introduction to linked data and its lifecycle on the web. In *Reasoning Web. Semantic Technologies for Intelligent Data Access*. Springer, 1–90.

[2] Ron Avnur and Joseph M. Hellerstein. 2000. Eddies: Continuously Adaptive Query Processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD '00)*. ACM, New York, NY, USA, 261–272.

[3] Surajit Chaudhuri. 1998. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (PODS '98)*. ACM, 34–43. DOI:http://dx.doi.org/10.1145/275487.275492

[4] Richard L. Cole and Goetz Graefe. 1994. Optimization of Dynamic Query Evaluation Plans. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data (SIGMOD '94)*. ACM, New York, NY, USA, 150–160.

[5] Pierre Courrieu. 2008. Fast computation of Moore-Penrose inverse matrices. *arXiv preprint arXiv:0804.4809* (2008).

[6] Goetz Graefe. 1993. Query Evaluation Techniques for Large Databases. *ACM Comput. Surv.* 25, 2 (June 1993), 73–169. DOI:http://dx.doi.org/10.1145/152610.152611

[7] Yannis E. Ioannidis, Raymond T. Ng, Kyuseok Shim, and Timos K. Sellis. 1992. Parametric Query Optimization. In *Proceedings of the 18th International Conference on Very Large Data Bases (VLDB '92)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 103–114.

[8] R. Isele, A. Jentzsch, and C. Bizer. 2011. Efficient Multidimensional Blocking for Link Discovery without losing Recall. In *WebDB*.

[9] Zachary G. Ives, Daniela Florescu, Marc Friedman, Alon Levy, and Daniel S. Weld. 1999. An Adaptive Query Execution System for Data Integration. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD '99)*. ACM, New York, NY, USA, 299–310. DOI:http://dx.doi.org/10.1145/304182.304209

[10] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *PVLDB* 3, 1 (2010), 484–493.

[11] Jorge J Moré. 1978. The Levenberg-Marquardt algorithm: implementation and theory. In *Numerical analysis*. Springer, 105–116.

[12] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. 2012. Eagle: Efficient active learning of link specifications using genetic programming. In *The Semantic Web: Research and Applications*. Springer, 149–163.

[13] Axel-Cyrille Ngonga Ngomo. 2012. On Link Discovery using a Hybrid Approach. *Journal on Data Semantics* 1, 4 (December 2012), 203 – 217.

[14] Axel-Cyrille Ngonga Ngomo. 2014. HELIOS - Execution Optimization for Link Discovery. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*. Springer, 17–32.

[15] Andriy Nikolov, Mathieu d'Aquin, and Enrico Motta. 2012. Unsupervised learning of link discovery configuration. In *9th Extended Semantic Web Conference (ESWC 2012)*. http://oro.open.ac.uk/33434/

[16] Xing Niu, Shu Rong, Yunlong Zhang, and Haofen Wang. 2011. Zhishi.links results for OAEI 2011. In *OM*.

[17] Muhammad Saleem, Maulik R Kamdar, Aftab Iqbal, Shanmukha Sampath, Helena F Deus, and Axel-Cyrille Ngonga Ngomo. 2014. Big linked cancer data: Integrating linked tcga and pubmed. *Web Semantics: Science, Services and Agents on the World Wide Web* 27 (2014), 34–41.

[18] Abraham Silberschatz, Henry Korth, and S. Sudarshan. 2006. *Database Systems Concepts* (5 ed.). McGraw-Hill, Inc., New York, NY, USA.

[19] Immanuel Trummer and Christoph Koch. 2014. Multi-objective Parametric Query Optimization. *Proc. VLDB Endow.* 8, 3 (Nov. 2014), 221–232. DOI:http://dx.doi.org/10.14778/2735508.2735512

[20] Tolga Urhan, Michael J. Franklin, and Laurent Amsaleg. 1998. Cost-based Query Scrambling for Initial Delays. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD '98)*. ACM, New York, NY, USA, 130–141. DOI:http://dx.doi.org/10.1145/276304.276317

[21] Chuan Xiao, Wei Wang, and Xuemin Lin. 2008. Ed-Join: an efficient algorithm for similarity joins with edit distance constraints. *Proceedings of the VLDB Endowment* 1, 1 (2008), 933–944.

[22] Chuan Xiao, Wei Wang, and Xuemin Lin. 2008. Ed-Join: An Efficient Algorithm for Similarity Joins with Edit Distance Constraints. *Proc. VLDB Endow.* 1, 1 (Aug. 2008), 933–944. DOI:http://dx.doi.org/10.14778/1453856.1453957

[23] Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. 2008. Efficient Similarity Joins for Near Duplicate Detection. In *Proceedings of the 17th International Conference on World Wide Web (WWW '08)*. ACM, New York, NY, USA, 131–140.