Performance Prediction for Graph Queries

Mohammad Hossein Namaki, Keyvan Sasani, Yinghui Wu, Assefaw H. Gebremedhin
Washington State University
{mnamaki, ksasani, yinghui, assefaw}@eecs.wsu.edu

ABSTRACT

Query performance prediction has shown benefits to query optimization and resource allocation for relational databases. Emerging applications are leading to search scenarios where workloads with heterogeneous, structure-less analytical queries are processed over large-scale graph and network data. This calls for effective models to predict the performance of graph analytical queries, which are often more involved than their relational counterparts.

In this paper, we study and evaluate predictive techniques for graph query performance prediction. We make several contributions. (1) We propose a general learning framework that makes use of practical and computationally efficient statistics from query scenarios and employs regression models. (2) We instantiate the framework with two routinely issued query classes, namely, *reachability* and *graph pattern matching*, that exhibit different query complexity. We develop modeling and learning algorithms for both query classes. (3) We show that our prediction models readily apply to resource-bounded querying, by providing a learning-based workload optimization strategy. Given a query workload and a time bound, the models select queries to be processed with a maximized query profit and a total cost within the bound. Using real-world graphs, we experimentally demonstrate the efficacy of our framework in terms of accuracy and the effectiveness of workload optimization.

1. INTRODUCTION

Graph queries have found prevalent use in knowledge extraction, traffic analytics, Web mining, social network analysis, and social media marketing. Common graph queries include (a) graph traversal, *e.g.,* reachability queries, and (b) pattern matching via subgraph isomorphism or simulation. Emerging applications require efficient workload processing with bounded resources [8]. Meanwhile, on real-world graphs that easily have billions of nodes and edges, graph queries are costly, even for reachability (linear-time), let alone subgraph isomorphism (NP-complete).

NDA'17 May 19, 2017, Chicago, IL, USA

DOI: http://dx.doi.org/10.1145/3068943.3068947

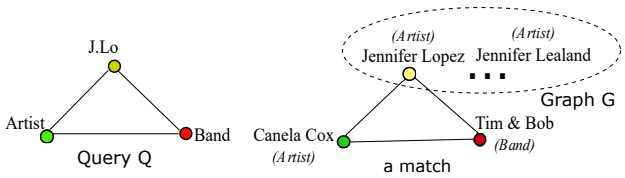


Figure 1: Approximate subgraph querying [19]

Database systems can greatly benefit from query performance prediction (QPP) for effective resource management and online query optimization. Given a workload $\mathcal{W}$, a database D, and performance metrics $\mathcal{M}$ (*e.g.,* response time, memory), the QPP problem is to predict $\mathcal{M}$ for each query instance in $\mathcal{W}$ over D. Various modeling and learning techniques for such predictions have been developed for relational queries [2,6,9–11]. These approaches, by employing statistics from relational algebra, relational data, and (logical and physical) query plans [2,12], have been found to be quite accurate for relational queries.

In contrast, QPP is particularly challenging for graph analytical workloads, for several reasons. First, graph queries, unlike their relational counterparts, can be "structureless" [29], *i.e.,* not well supported by rigid algebra and syntax. It is often hard to exploit algebra and operator-level features (*e.g.,* number of "join") [2,12] for graph queries (*e.g.,* reachability queries). Second, graph data is often noisy and heterogeneous. Features from data graph alone may not be reliable for QPP tasks. Third, while a common practice for QPP is to explore (logical and physical) query plans that are generated following a principled manner [2], this is inapplicable for approximate graph queries.

Example 1: Consider a query posed on a knowledge graph DBpedia that finds the artists who work with "J.Lo" in a Band [19]. This query can be represented by a graph pattern Q that carries (ambiguous) keywords, with a corresponding *approximate* match as illustrated in Fig. 1. Each pattern node in Q may have a large number of candidate matches.

The quality of the answer is usually determined only at run-time via similarity functions [29]. For example, "Canela Cox" is a best answer *when* "Jennifer Lopez" is matched to the ambiguous keyword "J.Lo" in the query. Conventional QPP that exploits relational algebra may not be applicable for graph pattern Q, as the syntax and declarative operators are hard to be derived from the "structureless" Q. Moreover, deriving statistics from the graph data alone is already expensive, due to the sheer size of data, and the fact that the underlying graph may change from time to time. □

In this paper, we present effective QPP methods for graph analytical workloads. We develop a practical learning paradigm that only makes use of computationally efficient query-oriented features and statistics from executed graph queries, without imposing any assumptions on graph data, query syntax and algebra. Our goal is to build a prediction framework for routinely issued, structureless graph queries.

Contributions. We make the following contribution.

(1) We propose a general learning framework to predict the query performance of graph analytical queries. The learning framework makes use of (a) three pragmatic classes of query-oriented features—called *query-*, *sketch-* and *algorithm-*features—that characterize query constraints, statistics of data to be accessed, and query evaluation behavior, respectively; and (b) well established regression models.

(2) Using the framework, we study two classes of frequently occurring graph queries and develop models and learning algorithms for each query class.

(3) We apply the query performance prediction to resource-bounded querying, and develop learning-based workload optimization strategies. The strategy aims to maximize the total profit of executed queries within a time bound, by selecting queries based on their predicted performance.

(4) Using real-world datasets, we experimentally verify the effectiveness of the query prediction and workload optimization. We find the following. (a) Our models can accurately predict the performance of graph queries utilizing conventional regression models and a handful of simple features. The best predictor attains an average accuracy of 0.93 and 0.96 over the knowledge graphs DBpedia and Freebase, respectively. (b) Our workload optimization strategy improves the effectiveness (measured by total profit of processed queries) of graph engines under bounded resource. On average, it achieves 92.78% of the effectiveness of an optimizer that knows the actual run-time of queries. These suggest useful QPP tools for effective online graph analytics.

The rest of the paper is organized as follows. In Section 2, we briefly introduce the graph query classes we consider as well as the algorithms used in our learning models as a black box. In Section 3, we describe the models that we have developed for prediction. In Section 4, we address workload optimization problem. In Section 5, we present experimental results. We describe related work in Section 6, and conclude in Section 7.

2. QPP FOR GRAPH QUERIES

2.1 Graph queries

Data graphs. We consider a labeled and directed data graph $G=(V, E, \mathcal{L})$, with node set V and edge set E. Each node $v \in V$ (edge $e \in E$) has a label $\mathcal{L}(v)$ ($\mathcal{L}(e)$) that specifies node (edge) information, and each edge represents a relationship between two nodes. In practice, $\mathcal{L}$ may specify attributes, entity types, and relation names [16].

Graph queries. We consider two familiar query classes.

Graph pattern queries. Graph pattern matching has been widely applied in areas such as social marketing, cyber security and knowledge extraction. A graph pattern Q is a graph (V_Q, E_Q, L_Q). Each *pattern node* $u \in V_Q$ has a label $L_Q(u)$ that describes the entities to be searched for (*e.g.,* type, attribute values), and an edge $e \in E_Q$ between two query nodes specifies the relationship posed on the two entities. A match of Q in G, denoted as $Q(G)$, is a subgraph of G that satisfies certain matching semantics. Given Q, G and a function $f(\cdot)$ that quantifies the quality of the match $Q(G)$ to Q, a *top-k graph pattern query* $Q(G, k, f)$ finds k best matches ranked by their quality as determined by $f(\cdot)$.

We consider two specifications for graph pattern queries (see Section 3.2) from approximate graph analytics:

- *Subgraph query* [29], which specifies a match $Q(G)$ as a subgraph induced by an isomorphism between Q and $Q(G)$; and
- *Simulation query* [17], which induces subgraphs $Q(G)$ with entities and relations that satisfy a (dual)-simulation relation between Q and $Q(G)$.

For both query classes, the quality of $Q(G)$ can be computed by aggregating (*e.g.,* summing up) a similarity score between each node (resp. edge) in Q and its counterpart(s) in $Q(G)$ by $f(\cdot)$ [29]. A subgraph query Q and its top match is illustrated in Fig. 1.

Reachability queries. Reachability queries are routinely used for traffic analysis, social analysis and Web mining. We consider a generalized reachability query $Q(G, s, t, d)$, where G is a graph, s is the source label, t is the target label, and d is a number. The query asks: "*Does there exist a source node with label s and a target node with label t that are connected by a path of length bounded by d in G?*" When $d{=}\infty$, and s and t each has a unique match, $Q(G, s, t, d)$ reduces to conventional single-source single-target reachability test.

Graph pattern queries and reachability queries are representative analytical queries. They do not have rigid query structures and syntax. Moreover, the query performance are determined at run-time, typically via a fixed-point computation. Conventional QPP that exploit operator- and plan-level statistics is not applicable for these queries.

In the next subsection, we describe query performance prediction for graph analytics. We address the query evaluation algorithms in Section 3, which are treated as "black-box" by our learning framework.

2.2 Query performance prediction

Our goal here is to formulate QPP for graph analytical workloads. We consider (1) a mixed workload $\mathcal{W}{=}\{Q_1, \ldots, Q_n\}$ over a set of query classes $\mathcal{Q}$, where each query Q_i is a query instance from a query class in $\mathcal{Q}$; and (2) response time t as the performance metric to be predicted. The problem of graph query performance prediction, denoted as GQPP, is to learn a prediction model $\mathcal{P}$ to predict the response time of each query instance in $\mathcal{W}$ with maximum accuracy measured by a specific metric.

Metric. A metric must measure how well the run-time of future queries are likely to be predicted. We seek to minimize the error depending on the type of the queries and the scale of their run-time. Therefore, we use *R-Squared*, a widely used evaluation metric [12] to evaluate our models. Given n graph queries, R-Squared (denoted as R^2) is computed as follows:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y_i})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where y, $\hat{y}$ and $\bar{y}$ are the actual values, predicted values, and the mean of actual values, respectively. The larger the R^2 value is (up to 1), the more accurate the model is.

To empirically verify the robustness of our models, we

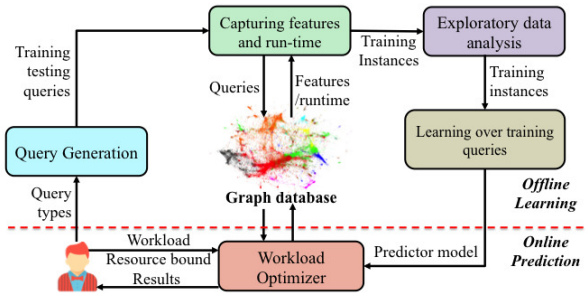


Figure 2: Learning framework for GQPP

also consider two other statistical metrics besides R^2: *normalized root mean square error* (NRMSE) and *mean absolute error* (MAE). NRMSE normalizes the standard deviation of the differences between predicted values and actual values (RMSE) normalized with standard deviation of actual values [27]. MAE, defined as $\frac{1}{n}\sum_{i=1}^{n}|\hat{y}-y|$, is an absolute comparison of predictions and eventual outcomes [10].

GQPP as regression. We approach GQPP as a regression problem. We use the following construction.

- **Input**: A data graph G, a query Q, a feature set $\mathcal{F}$
- **Output**: a prediction model $\mathcal{P}$ that maximize R^2

The problem is to learn, using regression, a function $f(x) = y$ that maps a feature vector x (that encodes the features in $\mathcal{F}$) to a continuous value y corresponding to the exact response time of the query. The goal is to minimize $|\hat{y}-y|$. That is, the closer $\hat{y}$ is to y, the closer R^2 is to 1.

3. PREDICTION FRAMEWORK

We next introduce our GQPP framework. We start with an overview of the framwork (Section 3.1). We then instantiate the general framework to graph pattern queries (Section 3.2) and reachability queries (Section 3.3).

3.1 Overview

The general GQPP framework is illustrated in Fig. 2. Following statistical learning, it derives a prediction model based on training sets (*i.e.,* offline learning), and then predicts query performance for unknown "test" data points based on the derived model (*i.e.,* online prediction). Our goal is then to adapt the framework to optimize the (mixed) graph analytical workload within a bounded resource.

Learning phase. This stage consists of the following. (1) The framework generates training workload $\mathcal{W}_T$ (to be discussed). To this end, it generates queries, evaluates the queries over the data graph G (stored and managed by graph database) by invoking standard query evaluation algorithms, and collects the performance metrics and features for each query to construct $\mathcal{W}_T$. (2) The predictive model is then derived by solving the regression problem (Section 2.2).

Features. As remarked earlier, graph analytical queries, unlike their relational counterparts, cannot be easily characterized by features from operators, algebra and apriori query plans. Features from data alone may also be unreliable. We hence consider three classes of query-oriented features. The features are called *(Q)uery*, *(S)ketch*, and *(A)lgorithm* features, since they characterize statistics from query instances, accessed data, and search behavior, respectively.

(a) *Query features* encode the topological constraints (*e.g.,* query size, degree, cyclic) and semantic constraints (*e.g.,* label, transformation functions [29]) from query terms.

(b) *Sketch features.* The idea is to exploit statistics that estimate the specificity and ambiguity of a query by "sketching" the data that will be accessed by the queries. These features may include the size of candidates (the nodes having the same or similar labels to some pattern query nodes), degree of sampled candidates, and statistics of sampled neighborhood of the candidates. By paying an affordable amount of time, these features significantly contribute to the prediction accuracy of graph queries (as verified in Section 5).

(c) *Algorithm features* refer to the features that characterize the performance of graph querying algorithms. For example, top-k graph search typically decomposes a query to subqueries, and assembles the complete results by aggregating partial matches as multi-way joins [28]. We found that features such as the number of decompositions and "joinable" candidates are very informative and critical to predict the cost of top-k search (see Section 3.2).

These features can be computed efficiently. Indeed, they can be extracted by fast linear scans and sampling over queries and data graphs, and are well supported by established database indexing techniques [28].

Training workload. Given a query class $\mathcal{Q}$, data graph G and a standard evaluation algorithm $\mathcal{A}$, the training workload $\mathcal{W}_T$ is a set of pairs $(Q_i.\mathcal{F}, t_i)$, where (1) Q_i is a query instance from $\mathcal{Q}$, $\mathcal{F}$ is a set of features, and $Q_i.\mathcal{F}$ refers to a feature representation of a query, and (2) t_i is the actual response time by evaluating Q_i with algorithm $\mathcal{A}$. Each pair is instantiated as a single data instance. Progressively, the data instances are appended to the training workload.

Predictive models. We evaluate a number of methods.

Linear regression. A simple start is linear regression, where the relationships of independent variables, *i.e.,* the features $Q_i.\mathcal{F}$, and the dependent variable, *i.e.,* the response time t_i of Q_i, are modeled using linear predictor functions. The goal is to find the coefficients a_k in the equation $t = a_1(Q.\mathcal{F}_1) + a_2(Q.\mathcal{F}_2) + ... + a_n(Q.\mathcal{F}_n) + \epsilon$ where ϵ is a noise term [9].

Regression trees. Linear regression may fail to accurately model dependencies that are beyond linear. We consider regression tree, a more complex model similar to a decision tree, except that it predicts continuous values instead of discrete classes.

Random forest. Significant improvements in predictors' accuracy have resulted from growing an ensemble of trees by asking them to vote for the predicted value [4]. Ensemble methods allow us to combine diverse weak regression trees by taking different samples of the original data set and then combining their outputs [22]. A random forest is a meta-estimator that fits some regression trees on various subsamples of the dataset and use averaging to improve the predictive accuracy and control over-fitting [20].

KNN. An alternative method is k-nearest neighbors (kNN) regression [1]. kNN predicts based on the k closest neighbors of an instance, determined by a predefined distance function. We consider Euclidean distance over the feature space.

SVM. We also consider SVM (Support Vector Machines) with nu-SVR kernel for regression [23] (SVM). SVM maps the features to a higher dimensional space to make it possible to perform the linear separation and perform the regression in that space.

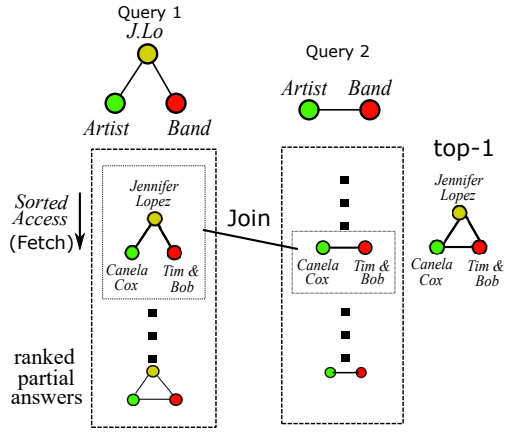


Figure 3: Illustration of top-k querying [19]

Correlation analysis. Exploratory data analysis (EDA) is an important step in data mining to understand the nature of the data. Leveraging summarization and visualization techniques, it seeks to identify critical features of a dataset and to eliminate outliers and anomalies [25]. We use EDA to compare the statistical characteristics of data sets (see Table 4), identify the relative importance of the features (see Tables 1, 2, and 3), and visually analyze the accuracy of predictors (Fig. 5(a)).

Online prediction. The prediction model is applied to predict the performance of newly arrived queries. Upon receiving a query workflow, the framework (1) collects the queries and computes the query features, and (2) predicts the query performance metrics. The predicted results can then be readily applied for resource allocation and workload optimization (see Section 4).

We next present the learning-based GQPP for specific graph analytical query classes. The major specification we need to make is to identify proper features.

3.2 GQPP for pattern queries

We consider two representative classes of graph pattern queries as follows.

Top-k Subgraph queries [28]. A top-k subgraph query $Q(G, k, f)$ defines a match using subgraph isomorphism, while f is derived by a set of functions drawn from a library (*e.g.,* acronym, synonym, abbreviations), where each function maps nodes and edges (as ambiguous keywords) in Q to their counterparts in G.

Algorithm. A common practice to evaluate $Q(G, k, f)$ is to follow the Threshold Algorithm [7] that aggregates top-k tuples in relational tables. Such an algorithm does the following (illustrated in Fig 3).

(1) Decomposes Q to n small fragments (subqueries, *e.g.,* stars, twigs) where partial answers can be efficiently evaluated. For each fragment, a sorted list is assigned.

(2) Iteratively generates and fetches partial matches for each fragment, and assembles the partial matches to complete ones whenever possible in a n-way join, following a sorted access to the lists in (1), until k complete answers are found or no new partial matches can be generated. Two optimizations are typically applied in step (2). (a) A "pivot" pattern node (*e.g.,* center of a star) is selected and assigned to each fragment of Q, so partial answers can be computed more efficiently. In accordance, a priority queue P is maintained for the pivot nodes, which bookkeeps their candidates with non-increasing order (determined by function f). (b) An estimated upper bound of the quality of the "unseen" answers and a lower bound of the quality of the "seen" complete matches are maintained and compared to guarantee early termination.

Table 1: Top features for subgraph queries

Feature	Type	Rel.imp.(%)
partial matches candidate	A	69.43
joinable nodes	A	15.24
pivots contributed in match(es)	A	8.45
node in fragments	A	5.45
pivot candidates	S	1.01
pivot degree avg.	S	0.22
# of query nodes	Q	0.09
# of query relationships	Q	0.07
# of fragments	A	0.03

Example 2: The algorithm finds top-1 answer for the query Q shown in Fig 1 as follows (illustrated in Fig 3). It first decomposes Q to two stars (Q_1 and Q_2). It then fetches partial answers for each star query in a sorted manner, and joins the partial answers whenever possible, until it finds a complete match and termination criteria is met. □

We use the algorithm in [28] as a "yardstick" for TA-style top-k subgraph search.

Features. We consider the following features.

(1) Query features (Q): indicates the query size, determined by the number of pattern nodes and edges. Intuitively, it usually takes more time to process larger queries.

(2) Sketch features (S): includes the number of candidates for pivot nodes. The more candidates pivot nodes have, the more expensive the join operations are.

(3) Algorithm features (A): Abusing the algorithm structure, we extract the features that are related to query decomposition behavior, such as number of decomposed sub-queries, number of nodes in each sub-query, and number of pivot nodes that are in a non-empty partial match, among others. The candidate size of pivot nodes can be efficiently estimated by using neighborhood indexing [28]. We also count the number of "joinable nodes" from the decomposition, *i.e.,* the number of pattern nodes shared by two sub-queries. The intuition is that the more joinable nodes there are, the more expensive it is to find a complete match due to the equi-join nature of the algorithm.

Observation. We report in Table 1 the most important features, ranked by their *relative importance* (determined by correlation and EDA component). In a decision tree, the importance of a feature $\mathcal{F}_i$ (mean decrease impurity) [15] is the total amount of impurity reduction by adding $\mathcal{F}_i$ to the tree, where impurity can be computed utilizing variance reduction [5]. In a forest model, the relative importance is averaged over all the trees for each feature [20].

We find that Algorithm and Sketch features play important role in predicting the performance of top-k subgraph queries. Query features, on the other hand, is less important. Indeed, the performance of top-k subgraph queries may highly depend on the algorithm behavior (decomposition, n-way joins in the TA-style computation), which can be more critical than the number of joins (a plan-level feature) in a graph pattern [12].

Table 2: Top features for Dual-simulation queries

Feature	Type	Rel. imp. (%)
sum of out-deg. in candidates	S	44.71
candidates of nodes	S	30.68
sum of in-deg. in candidates	S	24.18
# query edges	Q & A	0.22
# query nodes	Q	0.20

Dual-simulation [17]. A dual-simulation query $Q(G, k, f)$ relaxes the subgraph isomorphism from 1-1 bijective mapping to matching relations. Given $Q(V_Q, E_Q, L_Q)$ and graph $G(V, E, \mathcal{L})$, a match relation $R \subseteq V_Q \times V$ satisfies the following: (1) for any node $u \in V_Q$, there is a match $v \in V$ such that $(u, v) \in R$, and $L_Q(u)$=$\mathcal{L}(v)$; (2) for any $(u, v) \in R$, and any child (resp. parent) of u (denoted as u') in Q, there is a child (resp. parent) of v (denoted as v') in G, such that $(u', v') \in R$. That is, it preserves both parent and child relationships between a node u and its matches v.

As dual-simulation only considers label equality, f is a label equality function. It is known that there exists a unique, maximum matching relation R for dual-simulation [17]. Hence k=1 in $Q(G, k, f)$, and $Q(G, k, f)$ refers to the largest subgraph of G induced by the matches.

Algorithm. A standard algorithm for dual-simulation queries $Q(G, 1, f)$ starts by initializing, for each node u in Q, a candidate set $C(u)$ with the nodes having the same label. It then iteratively performs two join operations between the children of candidate sets $C(u)$ (resp. parents of $C(v)$) and $C(v)$ (resp. $C(u)$) for each edge $(u, v) \in E_Q$, and removes the candidates that cannot satisfy the constraints of dual-simulation by definition. This is repeated until a fixed point is reached (no candidate can be removed).

Features. Again, it is nontrivial to predict the performance of dual-simulation as a fixed-point computation. Luckily, most of the graph pattern queries are not large (*e.g.,* with diameters up to 2 [3]), and a proper "sketching" of the candidates may suggest accurate estimation. We consider the following features shown in Table 2 for dual-simulation, using the algorithm in [17] as yardstick algorithm.

(1) Query features: same as subgraph queries, we use number of nodes and edges of a query as query features.

(2) Sketch features: includes the size of candidates $|C(u)|$ for each pattern node u in a query Q and statistics of their degree information. Intuitively, more candidates indicate more expensive "joins" in the fixed-point computation. As dual-simulation preserves both parent and child relationship of each pattern node, we extract aggregations of both in-degrees and out-degrees of their candidates.

(3) Algorithm features: we consider the number of pattern edges as an algorithm feature as well, due to the nature of the fixed-point computation: it always follows query edges to perform join operations.

Observation. Unlike top-k subgraph queries, we find that Sketch features dominate the prediction of the efficiency of the computation for dual-simulation queries (as illustrated in Table 2). Candidate size and degree are critical for predicting dual-simulation query performance. Query size, on the other hand, is less important.

3.3 GQPP for reachability queries

We next investigate reachability queries. We use a suitably modified Breadth-First Search algorithm to evaluate reachability queries $Q(G, s, t, d)$. The algorithm finds the candidate sets for source and target labels, and performs reachability tests between the candidate sets of s and t.

Table 3: Top features for reachability queries

Feature	Type	Rel. imp. (%)
hop bound d	Q	57.37
sum out-deg. of src. cand.	S	33.09
sum in-deg. of src. cand.	S	4.36
candidates of src.	S	2.12
sum in-deg. of dest. cand.	S	1.30
candidate of dest.	S	0.93
sum out-deg. of dest. cand.	S	0.81

Features. We use the following features (illustrated in Table 3) for reachability queries.

(1) Query features: includes the hop bound d. Intuitively, the larger d is, the more expensive the traversal algorithm is, due to the larger set of nodes to be visited.

(2) Sketch features: include both candidate size for source and target labels, and statistics of their degrees.

(3) Algorithm features: As the behavior of regular breadth first search is not distinguishable for different reachability query instances, we do not consider algorithm features.

Observation. For reachability queries, Query feature d and Sketch features on degrees of source and target candidates are the most important features. Indeed, the more candidates and more neighbors they have, and the more hops a query needs to visit, the more expensive the query is.

These observations are further verified in our experimental study for graph pattern queries (see Table 6, Section 5).

4. WORKLOAD OPTIMIZATION

Once the prediction models are derived, they can be readily applied for workload optimization. As an application, we introduce a query selection strategy for resource bounded computation based on our GQPP framework. To this end, we formalize a workload optimization problem in the context of resource-bounded querying [8].

Workload optimization. Consider a mixed query workload $\mathcal{W}$=$\{Q_1, \ldots, Q_n\}$ over a set of query classes $\mathcal{Q}$, where each query Q_i is an instance of a query class in $\mathcal{Q}$, and is associated with a profit p_i. Given a graph G, workload $\mathcal{W}$ and a time bound T, the problem is to find a set of queries $\mathcal{W}$' such that (a) the total time cost for evaluating $\mathcal{W}$' is bounded by T, and (b) the total profit $\sum_{Q_i \in \mathcal{W}'} p_i$ is maximized. In practice, the profit may refer to *e.g.,* query importance, priority or QoS profit [21].

Given the GQPP predictor, we present an optimization strategy as follows. (1) Upon receiving $\mathcal{W}$, it invokes the predictor to estimate the time cost t_i for each query Q_i. (2) It then selects a set of queries $\mathcal{W}' \subseteq \mathcal{W}$ by solving the knapsack problem via the following construction. It takes each query Q_i as an item, the predicted run-time t_i (resp. profit p_i) of Q_i as item weights (resp. profit), and the time bound T as the size of a knapsack. We can verify that this construction is an approximation preserving reduction [24] from workload optimization to knapsack problem. It then invokes a fully polynomial time approximation scheme for knapsack problem [24] that greedily selects the query with high profit. This finds $\mathcal{W}$' with approximation ratio $(1 - \epsilon)$ for error bound $\epsilon \in (0, 1)$, and in $O(|\mathcal{W}|^2)$ time.

5. EXPERIMENTS

Using two real-world knowledge graphs, we conduct three sets of experiments to evaluate the following: (1) Performance of different machine-learning predictors for GQPP; (2) Impact of the factors (*e.g.,* size of training data) on the accuracy of the predictors; and (3) Effectiveness of query workload management, using a case study.

Experimental Setting. We used the following setting.

Datasets. The two real-world knowledge graphs we use for our experiments are Freebase and DBpedia. Freebase is a collaboratively created large knowledge base, containing 40.3M entities (e.g., people, companies, cities), 180M relationships, and 20K node and edge labels, extracted from several public knowledge bases including Wikipedia. DBpedia consists of 4.86M labeled entities (where each label is one of the 1K labels such as "Place", "Person", "Building") and 15M edges.

Workload. We develop two query generators, one for graph pattern queries (including subgraph queries and dual-simulation queries) and the other for reachability queries. These queries are used to construct the training and test sets over the two real-world knowledge graphs.

Graph pattern queries. To generate graph pattern queries $Q(k, f, G)$, we use the DBPSP benchmark [18], a DBpedia query benchmark. To achieve this, we first generate a set of query templates. Each template has a topology sampled from a graph category[1] as unlabeled graph. It is then assigned with a type sampled from the top 20% most frequent types in the ontologies of DBPedia and Freebase. We create 20 templates to cover common entity types and generate a total of 1K queries by instantiating the templates. The generated queries were used for both subgraph queires and dual-simulation queries. We draw the matching function f from a library of similarity functions as in [29]. We then set an integer k drawn from $[10, 100]$.

Reachability queries. For reachability queries $Q(s, t, d, G)$, we set $d \in [1, 4]$, and randomly select a pair of labels, from the top 20% most frequent labels in G. We sampled 4K queries, 1K for each d.

We execute each of these queries 5 times and record the average execution time in milliseconds (ms) for each query. Table 4 shows the average, minimum, and maximum running times of the query classes for our datasets. As the table shows, we have a mix of long and short running queries.

Algorithms. We implemented the following, all in Java.

(1) Standard query evaluation algorithms (Sec. 3), including: (a) STAR, the algorithm of [28] for top-k subgraph queries, (b) dual-simulation [17], for dual-sim queries, and (c) a variant of Breath-First Search, for reachability queries.

(2) Query workload optimization algorithms, including (a) our workload optimization algorithm Opt_RF, (b) Opt_Rnd, a baseline algorithm that randomly selects a next query to be executed in the workload, until the total response time reaches the time bound T, and (c) Opt_True, a counterpart of Opt_RF that uses the actual query response times to approximate the best solution.

Predictive models. We implemented all the methods (linear regression LR, regression tree RT, random forest RF, k-nearest neighbors kNN [1] and SVM [23]) with scikit-learn library [20], using their default settings.

[1] graphclasses.org

Metrics. We use three metrics as remarked in Section 2.2: R-Squared, NRMSE, and MAE.

Test platform. We ran all of our experiments on a Linux machine powered by an Intel 2.30 GHz CPU with 64 GB of memory. Each test is repeated 5 times and the averaged results are reported.

Result overview. We summarize our findings below.

(1) Using the three classes of features and conventional regression models, the performance of analytical graph queries can be predicted quite accurately (**Exp-1**).

(2) The predictors achieve high accuracy without using too large training query sets and very complex models (**Exp-2**).

(3) Our case study verifies the effectiveness of our approach for the variety of query workload managements including the single query class, the mixture of the queries, and profits assignments (**Exp-3**).

We next introduce our findings in details.

Exp-1: Performance of predictive models. We first evaluate the accuracy of predictive models LR, RT, and RF, as well as kNN and SVM.

R-Squared Accuracy. To estimate the accuracy of the predictors, we randomly select 20% for validation and the remaining to train the predictor. We iterate the process for 5 times and the average results are reported.

We report the accuracy of LR, RT, and RF for the three query classes, in Fig 4(a), Fig 4(b), and Fig 4(c), respectively. In all the cases, RF outperforms the alternatives. It reaches an accuracy range between 0.86 and 0.99 (resp. 0.98 and 0.97) for dual-simulation (resp. subgraph queries) in DBpedia and Freebase respectively. For reachability queries, it achieves 0.90 accuracy over Freebase and 0.95 accuracy for DBpedia. The higher accuracy of RF is due to its inherent ensemble methods that lead to the generality and robustness of the learned predictors [4]. The relatively lower accuracy of LR is due to the non-linearity of the independent variables (*i.e.,* feature values) with respect to the response variable (*i.e.,* performance metric).

Further, we find the average accuracy of kNN ($k = 2$ from $[2-4]$) and SVM to be even lower: 0.72 and 0.48 respectively. We thus omitted the results of kNN and SVM.

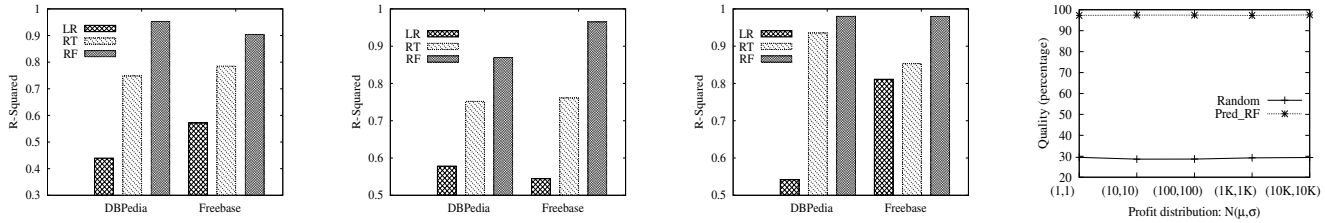
Alternative metrics. We report the average accuracy measured by the two alternative metrics MAE and NRMSE over Freebase and DBpedia in Table 5. We find the following. (1) In all cases, MAE is less than 0.8 seconds, which indicates a quite accurate prediction, especially for our query workload with a large variance of response time (as verified in Exp-3). Note that the accuracy of two different query types are not comparable using MAE, as it is scale-dependent. (2) NRMSE is in general small, especially for dual-simulation and top-k querying. These verify that the prediction models are quite robust to different metrics.

Actual vs. predicted. The comparison between predicted and actual execution times using RF for top-k querying over DBpedia is shown in Fig 5(a) for 1K queries.

We also find that the training of RF is quite efficient and feasible over large workload. It takes less than 1 second to train RF models over a training workload of 800 queries. Using well-supported graph neighborhood and label indices [29], it takes on average 15.3 seconds to predict

Table 4: Minimum, average, and maximum response time of the query samples

	Reacahbility			Dual-sim			Subgraph queries		
Dataset	Min (ms)	Avg (ms)	Max (ms)	Min (ms)	Avg (ms)	Max (ms)	Min (ms)	Avg (ms)	Max (ms)
DBPedia	5	592	12,553	8	16,828	60,015	122	6,754	63,582
Freebase	1	655	6,308	19	37,553	60,046	284	5,816	51,138

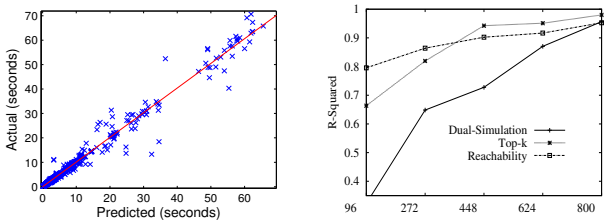


(a) Reachabilities Accuracy (b) Dual-Sim. Accuracy (c) Top-k Accuracy (d) Varying N(μ,σ)

Figure 4: Performance evaluation

Table 5: Accuracy of RF: alternative metrics

Dataset	Type	MAE (seconds)	NRMSE (%)
DBPedia	Reachability	0.163	14.02
	Dual-Sim.	0.800	14.98
	Top-k	0.271	6.43
Freebase	Reachability	0.177	23.21
	Dual-Sim.	0.111	1.25
	Top-k	0.130	4.20



(a) Actual vs. Prediction (b) Training Size

Figure 5: Accuracy of predictions

the performance of a workload of 433 queries, with total response time of 15 minutes.

Exp-2: Impact of factors. We next evaluate the impact of model factors on the accuracy of the best model RF.

Varying # of training queries. Fixing the size of test set at 200, we varied the number of training queries from 96 to 800, as shown in Fig 5 over DBpedia. The results tell us the following. (1) As expected, for all the query classes, RF gains a better accuracy over larger amount of training queries. (2) Not many training queries are needed to achieve a reasonable accuracy. For example, for the top-k querying, the accuracy is already above 0.90 after 448 queries. The results over Freebase (not shown) is consistent. In particular, R^2 is more than 0.85 for reachability after 272 queries and 0.88 after 448 queries for top-k querying.

Feature types. To further investigate the importance of the three type of features ((Q)uery, (S)ketch and (A)lgorithm), we trained seven predictors, one for a combination of the feature type for top-k queries over DBpedia. Table 6 summarizes the results. The results show the following. (1) Corresponding to the relative importance of the features in Table 1, for top-k, the query-only (resps. sketch) features perform poorly and the addition of those two does not add much to the accuracy of the results. (2) Algorithm-only features perform reasonably—they raise the accuracy to 0.925.

Table 6: Impact of feature type: Top-k (DBpedia)

Feature types	R-Squared
Q	0.268
S	0.329
A	0.925
Q+S	0.375
Q+A	0.942
S+A	0.930
Q+S+A	0.985

(3) The best accuracy (0.985) is obtained by combining all three categories of features.

Varying # of trees. We evaluated the impact of the number of estimators in RF, by varying the number of trees in RF from 1 to 100. As expected, by increasing the number of estimators until a saturated point, we observed better accuracy. In our experiments, no better performance is observed after 10 trees. Therefore, we fix the number of estimators to 10 for all of other experiments.

Exp-3: Query workload optimization. We next conduct case studies to test the effectiveness of GQPP for query workload optimization. To this end, we quantify the effectiveness of Opt_RF as $\frac{\sum_{Q_i \in \mathcal{W}} p_i}{\sum_{Q'_i \in \mathcal{W}'} p'_i}$, where $\mathcal{W}$ (resp. $\mathcal{W}$') refers to the selected queries of Opt_RF (resp. Opt_True). It measures the ratio of the total profits gained by Opt_RF to the total profits obtained by Opt_True, under time bound T. The effectiveness of Opt_Rnd is similarly defined.

We simulate workloads as follows. (1) We generate a workload of $6K$ queries, using a distribution 16%, 16%, and 68% for top-k subgraph, dual-sim and reachability queries, respectively. (2) We use absolute values of normal distribution $|N(\mu, \sigma)|$ with $(\mu, \sigma) = (1, 1)$ to generate profit of the queries. (3) The weighted queries are sent to each optimizer in batch. Given a bounded time, the optimizer selects queries to be executed.

Effectiveness of Opt_RF. We report the effectiveness of Opt_RF for the three reachability, dual-simulation, and top-k querying in Table 7, where the time bound is set to 15 and 30 minutes, respectively.

We observed the following. (1) In all experiments, the baseline algorithm Opt_Rnd fails to reach an effectiveness comparable to Opt_RF. (2) In average, Opt_RF outperforms Opt_Rnd by 5.1 for dual-simulation, by 2.75 for top-k querying, and by 2.23 for reachabilty. (3) Over both bounded

Table 7: Effectiveness of Opt_RF and Opt_Rnd

Dataset	Type	T (min)	Opt_Rnd (%)	Opt_RF (%)
DBPedia	Reachability	15	31.18	91.25
		30	43.72	95.76
	Dual-sim.	15	19.61	84.33
		30	28.95	89.32
	Top-k	15	23.32	97.98
		30	35.46	98.48
Freebase	Reachability	15	39.17	92.91
		30	53.89	95.49
	Dual-sim	15	7.57	89.72
		30	13.24	91.03
	Top-k	15	33.75	93.37
		30	46.65	93.79

times, Opt_RF only lost 4.09%, 6.14%, and 11.4% of the Opt_True profits for top-k querying, reachabilities, and dual-simulation respectively.

The source of Opt_RF error in compared with Opt_True is two-fold: (1) inference time of GQPP, and (2) the accuracy of prediction time. In the first sets of experiments, we have shown the inference time is negligible and the accuracy of prediction is high. Note that since knapsack problem is NP-Complete [24], both Opt_True and Opt_RF, using nearly the same amount of time, find an approximate solution.

Distribution of profits. We also evaluated the impact of profit distribution by varying μ and σ. Fixing the time bound to 15 minutes and a uniform mixture of query types, we varied $N(\mu, \sigma)$ from $(1, 1)$ to $(10K, 10K)$. Fig 4(d) tells us that Opt_RF remains to be effective, and is insensitive to the change of profits distribution. This verifies the robustness of the optimization strategy in terms of profit distribution.

Mixed query workload. We also evaluated the effectiveness of Opt_RF in the mixed workload. Fixing the time bound to 15 minutes, we created three sets of query workload over DBpedia, where each set is dominated by one query class. We report the effectiveness below.

Distribution	Opt_Rnd (%)	Opt_RF (%)
(70%,15%,15%)	23.93	95.60
(15%,70%,15%)	16.07	92.66
(15%,15%,70%)	23.12	95.59

As shown in the table, (1) The effectiveness of Opt_RF remains to be insensitive to the distribution of query classes; (2) When dual-simulation are dominant, we observe a slightly lower quality that is related to its lower predictor's accuracy. Moreover, we observe that dual-simulation takes more time than other query classes on average. Hence the impact of QPP predictor to the workload optimization becomes larger when it makes "mistakes".

6. RELATED WORK

Query performance prediction has been studied to estimate resource cost and effectiveness of query workloads [13]. We discuss some of this work below.

Relational queries. Learning techniques for QPP have been applied in relational databases for SQL workloads. The goal is to predict the response time and resource consumption of SQL queries leveraging: (1) calibrated cost models by analyzing a set of queries in offline, and adaptively refining the model units [26]; and (2) statistical machine-learning predictors constructed from several categories of features including plan-level [14], operator-level (*e.g.,* response time of operators in a parser tree), or both features [2]. The framework in [14] focused on learning to estimate CPU time and I/O costs of SQL query plans. In [9], a variation of Principal Components Analysis (PCA) used to predict the response time and other run-time characteristics of database requests. In [2], the authors advocated the use of support vector machines (SVM) as the specific machine-learning model. They further proposed a different approach by first building individual predictive models for each physical operator and then combining their predictions.

XML and SPARQL queries. QPP has been studied for querying semi-structured data, such as XML [30] and SPARQL [12]. Features are collected from XML parser trees, query plans, and topological of XML queries (*e.g.,* number of paths) to predict the response time of XML queries with regression models. Similarly, regression and Support Vector Machine are used to predict the performance of SPARQL queries, where the features are collected from SPARQL algebra and pattern [12]. In contrast to XML and SPARQL queries, graph analytical queries are not well supported by algebra and apriori query plans. These methods are not applicable to approximate graph querying scenarios.

Our work differs from previous work in several ways. (1) We study QPP for graph analytical queries, without assuming the existence of rigid syntax; (2) We exploit computationally efficient features that are collected by query input, run-time sketches and algorithms, which do not assume apriori query plans. Our experimental results over (mixed) graph analytical workloads verified the effectiveness of predictors, and suggests feasible QPP tools for online analytics.

7. CONCLUSION

We have presented a learning framework to predict response times of graph queries. We introduced learning methods for both graph pattern queries, defined by subgraph isomorphism and dual-simulation, and reachability queries. We show that by exploiting computationally efficient features from queries, sketches of the data to be accessed, and algorithm performance, the response times can be accurately predicted using regression models. We also introduced a workload optimization strategy for selecting the queries to be executed under bounded resources while maximizing profit. Our experimental study over real-world knowledge bases verifies the effectiveness of the learned predictors as well as the workload optimization strategy.

The study of QPP for complex graph analytical queries is in its infancy. We are testing our learning framework for more query classes that support graph mining and learning operators and algorithms. We plan to exploit features from query logs and user-feedback, for more performance metrics, including relevance, accuracy and the quality of the answers. Another interesting topic is to develop online machine learning techniques to support ad-hoc, self-tuning prediction over evolving query workload.

8. ACKNOWLEDGMENTS

Namaki and Wu are supported in part by NSF IIS-1633629 and Google Faculty Research Award. Sasani and Gebremedhin are supported in part by NSF CAREER award IIS-1553528.

9. REFERENCES

[1] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine learning*, pages 37–66, 1991.

[2] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik. Learning-based query performance modeling and prediction. In *ICDE*, 2012.

[3] M. Arias, J. D. Fernndez, M. A. Martnez-Prieto, and P. de la Fuente. An empirical study of real-world sparql queries. 2011.

[4] L. Breiman. Random forests. *Machine learning*, pages 5–32, 2001.

[5] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. 1984.

[6] J. Duggan, U. Cetintemel, O. Papaemmanouil, and E. Upfal. Performance prediction for concurrent database workloads. In *SIGMOD*, 2011.

[7] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66(4):614–656, 2003.

[8] W. Fan, X. Wang, and Y. Wu. Querying big graphs within bounded resources. In *SIGMOD*, 2014.

[9] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. Jordan, and D. Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *ICDM*, 2009.

[10] Q. Guo, R. W. White, S. T. Dumais, J. Wang, and B. Anderson. Predicting query performance using query, result, and user interaction features. In *Adaptivity, Personalization and Fusion of Heterogeneous Information*, 2010.

[11] C. Gupta, A. Mehta, and U. Dayal. Pqr: Predicting query execution times for autonomous workload management. In *Autonomic Computing, 2008. ICAC'08. International Conference on*, 2008.

[12] R. Hasan and F. Gandon. A machine learning approach to sparql query performance prediction. In *WI-IAT*, 2014.

[13] C. Hauff, D. Kelly, and L. Azzopardi. A comparison of user and system query performance predictions. In *CIKM*, 2010.

[14] J. Li, A. C. König, V. Narasayya, and S. Chaudhuri. Robust estimation of resource consumption for sql queries using statistical techniques. *Proc. VLDB Endow.*, pages 1555–1566, 2012.

[15] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts. Understanding variable importances in forests of randomized trees. In *NIPS*, pages 431–439, 2013.

[16] J. Lu, C. Lin, W. Wang, C. Li, and H. Wang. String similarity measures and joins with synonyms. In *SIGMOD*, 2013.

[17] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo. Capturing topology in graph pattern matching. *VLDB*, pages 310–321, 2011.

[18] M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. *DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data*, pages 454–469. 2011.

[19] M. H. Namaki, R. R. Chowdhury, M. R. Islam, J. R. Doppa, and Y. Wu. Learning to speed up query planning in graph databases. In *ICAPS*, 2017.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, pages 2825–2830, 2011.

[21] H. Qu and A. Labrinidis. Preference-aware query and update scheduling in web-databases. In *ICDE*, pages 356–365, 2007.

[22] G. Seni and J. F. Elder. Ensemble methods in data mining: improving accuracy through combining predictions. *Synthesis Lectures on Data Mining and Knowledge Discovery*, pages 1–126, 2010.

[23] S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy. Improvements to the smo algorithm for svm regression. *TNNLS*, pages 1188–1193, 2000.

[24] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[25] P. F. Velleman and D. C. Hoaglin. *Applications, basics, and computing of exploratory data analysis*. 1981.

[26] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacigümüs, and J. F. Naughton. Predicting query execution time: Are optimizer cost models really unusable? In *ICDE*, pages 1081–1092, 2013.

[27] K. Yang, J. Li, and C. Wang. Missing values estimation in microarray data with partial least squares regression. In *ICCS*, pages 662–669, 2006.

[28] S. Yang, F. Han, Y. Wu, and X. Yan. Fast top-k search in knowledge graphs. 2016.

[29] S. Yang, Y. Wu, H. Sun, and X. Yan. Schemaless and structureless graph querying. *VLDB*, 2014.

[30] N. Zhang, P. J. Haas, V. Josifovski, G. M. Lohman, and C. Zhang. Statistical learning techniques for costing xml queries. In *VLDB*, pages 289–300, 2005.