Select-and-Evaluate: A Learning Framework for Large-Scale Knowledge Graph Search

F A Rezaur Rahman Chowdhury‡ FCHOWDHU@EECS.WSU.EDU
Chao Ma† ‡ MACHAO@EECS.OREGONSTATE.EDU
Md Rakibul Islam MISLAM1@EECS.WSU.EDU
Mohammad Hossein Namaki MNAMAKI@EECS.WSU.EDU
Mohammad Omar Faruk MFARUK@EECS.WSU.EDU
Janardhan Rao Doppa JANA@EECS.WSU.EDU
School of EECS, Washington State University, Pullman, WA 99164, USA
†*School of EECS, Oregon State University, Corvallis, OR 97331, USA*
‡*Equal contribution from first two authors*

Editors: Yung-Kyun Noh and Min-Ling Zhang

Abstract

Querying graph structured data is a fundamental operation that enables important applications including knowledge graph search, social network analysis, and cyber-network security. However, the growing size of real-world data graphs poses severe challenges for graph search to meet the response-time requirements of the applications. To address these scalability challenges, we develop a learning framework for graph search called **S**elec**t**-and-Ev**al**ua**t**e (SCALE). The key insight is to select a small part of the data graph that is sufficient to answer a given query in order to satisfy the specified constraints on time or accuracy. We formulate the problem of generating the candidate subgraph as a computational search process and induce search control knowledge from training queries using imitation learning. First, we define a search space over candidate selection plans, and identify target selection plans corresponding to the training queries by performing an expensive search. Subsequently, we learn greedy search control knowledge to imitate the search behavior of the target selection plans. Our experiments on large-scale knowledge graphs including DBpedia, YAGO, and Freebase show that using the learned selection plans, we can significantly improve the computational-efficiency of graph search to achieve high accuracy.

Keywords: Knowledge Graphs, Structured Prediction, Learning for Search

1. Introduction

Graph representations are employed for modeling data in many real-world applications including knowledge graphs, social and biological networks. Graph querying is the most primitive operation for information access, retrieval, and analytics over graph data that enables applications including knowledge graph search, and cyber-network security. We consider the problem of graph search, where the input is a *data graph* and a *graph query*, and the goal is to find the answers to the given query by searching the data graph. For example, to detect potential threats, a network security expert may want to "find communication patterns matching the attack pattern of a newly discovered Worm" over a cyber network (3). This natural language query is issued using a formal graph query language like SPARQL. Specifically, we study the graph querying problem as illustrated in Example 1.

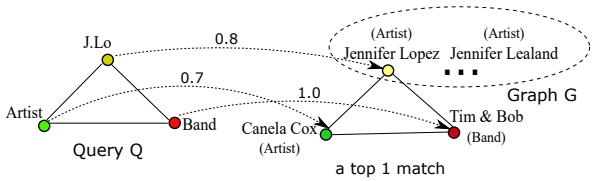


Figure 1: Illustration of the graph querying problem (33).

Example 1. Consider a graph query Q on DBpedia knowledge graph, shown in Figure 1. We want to find the artists who work with "J.Lo" in a Band. Each of these ambiguous nodes in Q can have excessive number of candidate matches. For example, there are 31,771 nodes with label "Band" and 82,853 "Artists" in DBpedia. In addition, "J.Lo" can match to different people whose first name starts with "J" or lastname starts with "Lo". While "Jennifer Lopez" is the perfect match for this query, users may want to get other similar people like "Jennifer Hudson", etc. However, the matching score for each candidate node can be different due to different node context. It is computationally expensive to first expand all of those matches to find the query structure and then rank them based on their matching scores.

The growing size of real-world data graphs in emerging real-world applications poses severe challenges. For example, knowledge graphs employed in health-care and enterprise applications, and cyber networks employed in network monitoring applications may involve billions of linked entitites with rich associated information. Therefore, performing even a linear scan over these large graphs is not possible within the real-time requirements of the application (e.g., network intrusion detection and defense). Additionally, common graph queries are computationally expensive, e.g., evaluation of subgraph isomorphism query is NP-hard. As we overview in the next section, existing graph search algorithms follow the "one-size-fits-all" philosophy: all queries are processed via predefined search strategies over entire data graph for every application. The main drawbacks of existing algorithms include: 1) Query evaluation techniques scale poorly for large-scale data graphs; 2) Predefined search procedures lack the flexibility to meet the requirements/constraints of the applications (e.g., finding answers to a query within a time-bound); and 3) Cannot leverage the knowledge of query distribution to automatically optimize the search procedure.

A primary contribution of this paper is to address the above challenges and drawbacks by studying a learning framework that allows to trade-off speed and accuracy of graph search based on the needs of the application. This problem can be seen as an instance of learning to speed up structured prediction (17; 41; 11; 13; 28; 12). The general idea is to specify a computationally-bounded search architecture with some parameters, and automatically optimize its performance based on the needs of the application using training queries drawn from a target distribution. Specifically, we develop a framework called **S**ele**c**t-and-Ev**al**uat**e** (SCALE) to efficiently search large data graphs. The key insight is to *select a small part of the data graph that is sufficient to answer a given query in order to satisfy the specified constraints on time or accuracy*, which is loosely inspired by the $\mathcal{HC}$-Search framework (11). Given a query, data graph, and search bound τ as input, the framework first selects a candidate subgraph of size τ to answer the input query and subsequently, evaluates the input query on the selected candidate subgraph to find the best answer. The problem of generating the candidate subgraph is formulated as a computational search process guided by a learned policy. The role of policy is to quickly guide the search towards high-quality candidate subgraphs.

We develop a *imitation learning* methodology for inducing greedy policies using training queries drawn from a target distribution. First, we define a concrete search space over candidate selection plans, where each selection plan can be seen as a sequence of actions to progressively construct a subgraph of size τ. The quality of a candidate selection plan for a given graph query is defined as the accuracy of the answer produced by evaluating the given query on the corresponding subgraph. Second, we perform a computationally-expensive search (heuristic guided beam search) in this search space, to identify target selection plans corresponding to the training queries. Third, we learn greedy policies in the framework of imitation learning to mimic the search behavior of these target selection plans to quickly generate high-quality candidate subgraphs. Our experiments on large-scale knowledge graphs including DBpedia, YAGO, and Freebase show that SCALE framework with the learned policy can significantly improve the computational-efficiency of graph search to achieve high accuracy.

Contributions. The main contributions this paper are as follows:

1. A novel learning framework called **S**ele**c**t-and-Ev**al**uat**e** (SCALE) for large-scale graph search is developed. The key idea is to select a small candidate subgraph that is approximately sufficient to answer a given graph query (Section 3).

2. Designing a imitation learning methodology for inducing greedy policies to guide the search for high-quality candidate subgraph. This includes specifying a concrete search space, constructing a oracle policy, and defining appropriate features to drive the learning process (Section 5).

3. Comprehensive evaluation of the SCALE framework and its variants on three large real-world knowledge graphs, namely, YAGO, DBpedia, and Freebase (Section 6).

2. Background and Prior Work on Graph Query Processing

In this section, we provide the background on graph query processing, and the related work on top-k graph search including the limitations of the state-of-the-art approaches.

Data Graph. 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 heterogeneous attributes, entities, and relations (29).

Graph Query. We consider query Q as a graph (V_Q, E_Q, L_Q). Each *query node* in Q provides information/constraints about an entity, and an edge between two nodes specifies the relationship posed on the two nodes. Formal graph query languages including SPARQL (36), Cypher, and Gremlin can be used to issue graph queries to a database such as Neo4j(neo4j.com). Since existing graph query languages are essentially subgraph queries, we can invoke a query transformer to work with different query languages (27). Therefore, our work is general and is not tied to any specific query language.

Subgraph Matching. Given a graph query Q and a data graph G, a match of Q in G is a mapping $\phi \subseteq V_Q \times V$, such that: 1) ϕ is an injective function, *i.e.,* for any pair of distinct nodes u_i and u_j in V_Q, $\phi(u_i) \neq \phi(u_j)$; and 2) for any edge $(v_i, v_j) \in E_Q$, $(\phi(v_i), \phi(v_j)) \in E$. The match $\phi(Q)$ is a *complete* match if $|Q| = |\phi(Q)|$, where $|Q|$ denotes the size of Q, *i.e.,* the sum of the number of nodes and edges in Q (similarly for $|\phi(Q)|$). Otherwise, $\phi(Q)$ is a *partial match*.

Matching Score. Given Q and a match $\phi(Q)$ in G, we assume the existence of a scoring function $F(\cdot)$ which computes, for each node $v \in V_Q$ (resp. each edge $e \in E_Q$), a matching score $F(\phi(v))$

(resp. $F(\phi(e))$). The matching score of ϕ is computed by a function $F(\phi(Q))$ as

$$F(\phi(Q)) = \sum_{v \in V_Q} F(v, \phi(v)) + \sum_{e \in E_Q} F(e, \phi(e)) \quad (1)$$

One can use any similarity function such as acronym, abbreviation, edit distance etc. We adopt Levenshtein function (34) to compute node/edge label similarity, and employ the R-WAG's ranking function that incorporates both label and structural similarities (38) to compute matching score $F(\cdot)$.

Top-k Graph Querying. Given Q, G, and $F(\cdot)$, the top-k subgraph querying problem is to find a set of k matches $Q(G,k)$, such that for any match $\phi(Q) \notin Q(G,k)$, for all $\phi'(Q) \in Q(G,k)$, $F(\phi'(Q)) \geq F(\phi(Q))$.

Top-k Search Algorithms. Top-k graph querying problem is known to be intractable (NP-hard). The common practice in existing top-k graph search techniques (39; 10; 2; 43; 40; 45; 42) is to follow a conventional Threshold Algorithm (TA) style approach (14). TA framework consists of three main steps: **Step 1:** *Query Decomposition.* A query Q is decomposed into a set of sub-queries $\{Q_1, \ldots, Q_T\}$ without loss of generality. The procedure then initializes one list for each sub-query to store the partial answers. For relational databases, sub-queries can correspond to individual attributes (14); and for graph queries, sub-queries can correspond to edges (39; 45; 2; 10) or stars (42); **Step 2:** *Partial Answer Generation.* For each sub-query Q_i, it performs an iterative exploration as follows. a) It computes and fetches k partial matches of Q_i; and b) It verifies if the partial matches can be joined with other "seen" partial matches to be able to improve the top-k matching score; and **Step 3:** *Early Termination.* TA dynamically maintains a) an lower bound (LB) as the smallest top-k match score so far; and b) an upper bound (UB) to estimate the largest possible score of a complete match from unseen matches. If $UB < LB$, TA terminates.

STAR (42) is the state-of-the-art graph search algorithm: instantiation of the TA framework, where each sub-query takes the form of a star (i.e., graph with a unique pivot and some adjacent nodes). It was shown both theoretically and empirically that star decomposition provides a good trade-off between sub-query evaluation cost and the number of candidate partial answers. The main limitations of TA framework and STAR are as follows: 1) Applying fixed search strategy may not provide robust performance due to diversity in queries; 2) The performance of TA framework highly depends on a good estimation of the upper bound; and 3) It is hard to adapt TA framework to different *resource constraints* (e.g., memory, response time, accuracy, I/O cost).

3. SCALE Framework

In this section, we first provide a high-level overview of our SCALE framework. Subsequently, we describe the details of both selection and evaluation components.

Key Idea. Our SCALE framework is an instantiation of the following general idea. We accept that graph search is a NP-hard problem. Therefore, we specify a computationally-bounded search architecture with some parameters, and automatically optimize its performance based on the needs of the application using training queries drawn from a target distribution.

Overview of the SCALE Framework. There are two main computational components in SCALE: 1) *Selection*, whose role is to generate a candidate subgraph of size τ (parameter) to answer the input query; and 2) *Evaluation*, whose role is to find the best answer by evaluating the input graph query on the generated subgraph. The scoring function $F(.)$ is used to score the candidate answers. Given a graph query Q, data graph G, and search bound τ, the selection component generates a

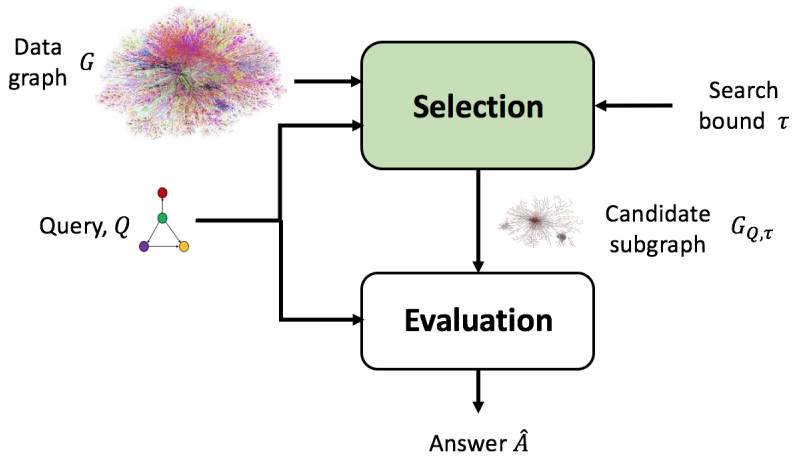


Figure 2: High-level overview of the SCALE framework.

subgraph $G_{Q,\tau} \subset G$ whose size is τ and the evaluation component finds the best answer $\hat{A}$ by searching $G_{Q,\tau}$.

Selection Component. The selection component performs state space search and consists of two key elements: 1) *Search Space* $\mathcal{S}_p = (I, A)$ over candidate subgraphs of the data graph G that can be instantiated for a given query Q, where $I(Q)$ is the initial state and $A(s)$ refers to the available actions at a state s. We will provide a concrete search space definition as part of our learning methodology; and 2) *Selection Policy* Π that guides the greedy search from $I(Q)$ until it uncovers a subgraph of size τ. Note that it is computationally intractable to perform exhaustive search over all candidate subgraphs of size τ.

For a given graph query Q, the search starts with an empty graph (initial state s_0); and grows the partial graph based on the set of possible actions A (e.g., adding an edge) and the greedy policy Π that selects the best scoring action at each state. The search is continued until a candidate subgraph of size τ is found. The corresponding subgraph $G_{Q,\tau}$ will be selected for evaluation.

Evaluation Component. Given a query Q, the corresponding selected subgraph $G_{Q,\tau}$, and a scoring function $F(.)$ to score the candidate answers, the evaluation algorithm needs to search for the best scoring candidate answer $\hat{A}$. This corresponds to solving an instance of subgraph isomorphism

Algorithm 1 SCALE Framework

Input: Q = graph query, G = data graph, F = scoring function, (I, A) = search space, τ = search bound, Π = selection policy

1: $s_0 \leftarrow I(Q)$ *// Initial state with empty subgraph*
2: **for** each selection step t=1 to τ **do**
3: $\quad a_{best} \leftarrow \arg max_{a \in A(s_{t-1})} \Pi(s, a)$ *// Pick the best scoring action*
4: $\quad s_t \leftarrow$ Apply a_{best} on s_{t-1} *// Grow the subgraph*
5: **end for**
6: $G_{Q,\tau} \leftarrow$ subgraph from state s_τ *// Subgraph of size τ selected by greedy policy Π*
7: $\hat{A} \leftarrow$ `Evaluate`$(Q, G_{Q,\tau}, F)$ *// Evaluation of query Q on the selected subgraph $G_{Q,\tau}$*
8: **return** answer $\hat{A}$

problem. We will employ the following two algorithms in our experiments: VF2 (exact and expensive) and Dual Simulation (approximate and cheap).

1) VF2 Algorithm. Cordella et al. proposed an exact algorithm named *VF2* to solve subgraph isomorphism (4; 5). The VF2 algorithm considers a state space search formulation, where states correspond to a partial mapping solution between Q and $G_{Q,\tau}$. The search space is explored in a depth-first manner guided by a fixed set of pruning rules to make the search efficient. The time complexity is $\Theta(N)$ and $\Theta(N! \cdot N)$ in the best-case and worst-case respectively, where N is the total number of nodes in Q and $G_{Q,\tau}$.

2) Dual Simulation Algorithm. Simulation and dual simulation allows to capture meaningful, approximate matches for graph patterns by relaxing the strict "one-to-one" mapping requirement by subgraph isomorphism (15; 31). Intuitively, a dual simulation based pattern matching preserves the path information between query and the corresponding match. The time complexity of the state-of-the-art dual simulation algorithm for query Q and selected subgraph $G_{Q,\tau}$ is $\mathcal{O}(N_Q \cdot N_{G_{Q,\tau}})$ (31), where N_Q and $N_{G_{Q,\tau}}$ correspond to the total number of nodes and edges in Q and $G_{Q,\tau}$ respectively.

4. Problem Setup

We assume the availability of a data graph G and a training set of queries $\mathcal{T} = \{Q\}$ drawn from an unknown target distribution D, where Q is a graph query. Unlike traditional supervised learning problems, we do not know the correct answers for these training queries (recall that graph search is a NP-Hard problem). For a given query Q, search bound parameter τ (total number of nodes and edges in Q), and candidate subgraph $G_{Q,\tau} \subset G$, suppose a query evaluation algorithm `Evaluate` produces $\hat{A}$ as the corresponding answer. We define the *pessimistic* loss function L as follows:

$$L(Q, \hat{A}) = 1 - \frac{F(Q, \hat{A})}{\texttt{SMAX(Q)}} \tag{2}$$

, where `SMAX(Q)` is the maximum possible score according to $F(.)$ for a candidate answer of query Q. This loss function is pessimistic because even if we run an exact evaluation algorithm (e.g., VF2) on query Q and data graph G, the best answer A^* may have a score less than `SMAX(Q)`.

We are given a search space definition $\mathcal{S}_p = (I, A)$ over candidate subgraphs of the data graph G that can be instantiated for a given query Q and a search bound parameter τ. We focus on greedy search (see Algorithm 1). For a given input query Q, the decision process for producing the subgraph $G_{Q,\tau}$ corresponds to choosing a sequence of actions leading from the initial state using the selection policy Π, until the search process uncovers a subgraph of size τ. In this work, we assume a linear function to represent the selection policy Π: $\Pi(s, a) = w \cdot \Psi(s, a)$, where $\Psi(s, a) \in \Re^m$ represents the features of a state action pair (s, a) (i.e., query Q and the subgraph formed due to action a) and $w \in \Re^m$ correspond to the parameters of the policy. We want to learn the parameters of Π using the training queries $\mathcal{T}$, with the goal of minimizing the loss on unseen queries drawn from the target distribution.

5. Learning Methodology

In this section, we explain the different components of our imitation learning methodology that includes defining a concrete search space, constructing a good oracle policy, and defining appropriate features to drive the learning process.

5.1. Search Space

We first explain the building blocks that we will use to define our search space, and then define a concrete search space over candidate subgraphs of the data graph G for a given query Q.

Building Blocks. We need the following two elements to define an effective search space. **1) Seed Nodes:** For a input query Q with vertex set V_Q, we obtain a list of seed nodes `CAND(v)` for each node $v \in V_Q$ using the following scheme: We first compute the best-1 label match set based on node label similarity to $v \in V_Q$ using the Levenshtein function (34). Subsequently, we consider all the nodes in the data graph that contain the labels in that set as our seed nodes. For computational efficiency, we pre-compute these lists for all node labels in the data graph G and store them as an index. **2) One-Hop Index:** We also pre-compute a One-Hop index for all nodes in the data graph G to be able to quickly prune bad search paths. For each node u, we store two indices: `IN(u)` and `OUT(u)` that represents the labels of the nodes that have an incoming edge to the node u and outgoing edge from the node u respectively.

Search Space Definition. The search space $\mathcal{S}_p = (I, A)$ is defined in terms of an initial state function I and action set function A that gives the available actions at any state to be able to generate the successor states. **1) Initial State Function:** For a input query Q, $I(Q)$ gives the initial state. The initial state corresponds to an empty graph. **2) Successor Function:** $A(s)$ refers to the available actions at a state s that corresponds to different ways of growing the subgraph at state s. If s contains an empty graph, the action set $A(s)$ represent adding different seed nodes of the vertex $v \in V_Q$ that has the smallest $|\texttt{CAND(v)}|$. For states with non-empty subgraph, we execute the following procedure to get the available actions $A(s)$: We define a depth-first search (DFS) based ordering over the edges E_Q of the query Q and perform a round-robin selection over these edges. For each selected edge $e \in E_Q$, we consider candidate matches to this edge (via One-Hop index) to grow the subgraph of the current state. If the selected edge does not have any more candidate matches to grow the subgraph, we move to the next edge in the DFS order.

5.2. Learning Search Control Knowledge

Inspired by the recent success of imitation learning approaches for solving sequential decision-making tasks (21; 23; 22; 35), we formulate and solve the problem of learning policies to select high-quality subgraphs in the framework of imitation learning.

Overview of Imitation Learning. In traditional imitation learning, expert demonstrations are provided as training data (e.g., demonstrations of a human expert driving a car), and the goal of the learner is to learn to imitate the behavior of an expert performing a task in a way that generalizes to similar tasks or situations. Typically this is done by collecting a set of trajectories of the expert's behavior on a set of training tasks. Then supervised learning is used to find a policy that can replicate the decisions made on those trajectories. Often the supervised learning problem corresponds to learning a classifier or regressor to map states to actions, and off-the-shelf tools can be used.

The two main challenges in applying imitation learning framework to selection plan generation are: 1) Obtaining a high-quality oracle policy that will provide the supervision for the imitation learning process (*oracle construction*); and 2) Learning search control policies that can make search decisions in a computationally-efficient manner (*policy learning*). We provide solutions for these two challenges below.

1) Oracle Construction. We describe a generic procedure to compute high-quality selection plans within the given search space $\mathcal{S}_p = (I, A)$. We will start by defining the needed terminology.

Definition 1. For a given query Q, a state s in the search space $\mathcal{S}_p$ is called a *terminal state* if the size of the subgraph corresponding to the state s is equal to τ, the search bound parameter.

Definition 2. For a given query Q, a sequence of actions from initial state to terminal state, $(s_0, a_0), \cdots, (s_\tau, \varnothing)$, is called a *selection plan* (SP), where s_0 is the initial state and s_τ is the terminal state.

Definition 3. For a given query Q, the *quality of a SP* is defined as the loss $L(Q, \hat{A}) \in [0, 1]$ (smaller the better), where $\hat{A}$ is the answer obtained by executing a query evaluation algorithm `Evaluate` using the corresponding subgraph $G_{Q,\tau}$.

The goal of oracle construction is to compute high-quality selection plans for each training query Q. A naive approach would be to perform depth-bounded exhaustive search in search space $\mathcal{S}_p$ instantiated for each training query. Since the naive approach is impractical, we resort to heuristic guided beam search and define a effective heuristic function to make the search efficient.

Heuristic Function. The heuristic function H takes a state s (query $Q = (V_Q, E_Q)$ and corresponding subgraph $G' = (V', E') \subset G$) and returns a numeric score (higher the better). We define our heuristic function as follows: $H(s) = \sum_{u \in V_Q} H_p(u)$, where H_p is a primitive heuristic function. The primitive heuristic H_p assigns a score to each node $u \in V_Q$ based on the candidate subgraph G':

$$H_p(u) = max_{v \in V'} S(u, v) + P(u, v) \tag{3}$$

, where $S(u, v)$ is the normalized similarity between u and v based on the Levenshtein function, and $P(u, v)$ is the normalized potential function that assigns scores to vertices $v \in V'$ proportional to the neighboring edges $N(v)$ that can potentially match with the neighboring edges $N(u)$.

Target Selection Plan Computation via Beam Search. For each training query Q, we perform best-first beam search with beam width b starting at initial state $s_0 = I(Q)$, until we uncover a terminal state (i.e., size of subgraph at the state is τ). The sequence of actions leading from s_0 to terminal state s_τ is identified as the target selection plan for Q (see Algorithm 2).

The quality of the selection plans produced by heuristic-guided beam search will depend critically on two parameters: 1) beam width b; and 2) search bound τ. Intuitively, the quality of plans will improve as the values of b and τ increase, although at the expense of additional computation. In practice, we will vary both b and τ to evaluate the quality of the oracle policy on all the training queries before starting the imitation learning process.

Algorithm 2 Target Selection Plan Computation

Input: Q = query, (I, A) = search space,
τ = search bound, b = beam width

$B \leftarrow s_0 = I(Q)$ // *Initial state*
TERMINATE $\leftarrow False$
while not TERMINATE **do**
  $s \leftarrow$ best scoring state in B // *Selection*
  $C \leftarrow \emptyset$ // *Candidate set*
  if size of subgraph at s == τ **then**
    TERMINATE $= True$ and $s_\tau = s$
  else
    Expand s and add all next states to C // *Expansion*
  end if
  $B \leftarrow$ Top-b scoring states in C via heuristic // *Pruning*
end while
return state-action pair sequence from s_0 to s_τ

2) Learning Greedy Policies via Imitation Learning. Our goal is to learn a greedy policy Π that maps states to actions in order to imitate the target selection plans computed via oracle construction. We assume that for any training query Q, we can get the oracle selection plan

$(s_0^*, a_0^*), (s_1^*, a_1^*), \cdots, (s_\tau^*, \varnothing)$ via Algorithm 2, where s_0^* is the initial state and s_τ^* is the terminal state. The goal is to learn the parameters of Π such that at each state s_i^*, $a_i^* \in A(s_i^*)$ is selected.

Exact Imitation Approach. At each state s_t^* on the target selection plan of a training query Q, we create a ranking example such that the score of the best action $a_t^* \in A(s_t^*)$ is higher than all other actions: $w \cdot \Psi(s_t^*, a_t^*) > w \cdot \Psi(s_t^*, a), \forall a \in A(s_t^*) \setminus a_t^*$. The sets of aggregate ranking imitation examples collected over all the training queries are then fed to a rank learner (e.g., Perceptron or SVM-Rank), to learn the parameters of Π (see Algorithm 3). This reduction allows us to leverage powerful off-the-shelf rank learners. In theory and practice, policies learned via exact imitation can be prone to error propagation: errors in the previous state contributes to more errors.

DAgger Algorithm. DAgger is an advanced imitation approach (37) that generates additional training data so that the learner is able to learn from its mistakes. It is an iterative algorithm, where each iteration adds imitation data to an aggregated data set. The first iteration follows the exact imitation approach. After each iteration, we learn policy Π using the current data. Subsequent iterations perform selection planning using the learned policy to generate a trajectory of states for each training query. At each decision along this trajectory, we add a new imitation example if the search decision of the learned policy is different from the oracle policy. In the end, we select the best policy over all the iterations via performance on validation data.

3) Features. Recall that we assume a linear function to represent the selection policy Π: $\Pi(s, a) = w \cdot \Psi(s, a)$, where $\Psi(s, a) \in \Re^m$ represents the features of a state action pair (s, a) (i.e., query Q and the subgraph formed due to action a). To drive the learning process, we define the following category of features for Ψ: 1) Number of candidate matches for each query node in the subgraph; 2) Number of candidate node matches for a query node in the subgraph that satisfy the one-hop neighborhood label index property of nodes in the query graph; 3) Same as 2 but in the data graph; 4) We perform a random walk of fixed size starting from each of the nodes in the query graph. We check from how many nodes in the subgraph, we can do the same random walk; 5) Size of the query graph, subgraph, and number of available actions to grow the subgraph from new state; 6) Number of candidates present for the last selected query edge and also for each of the nodes of the last selected query edge. Alternatively, as mentioned in section 6.3, we can employ a deep learning model to learn appropriate representation automatically.

Algorithm 3 Learning Greedy Policy via Exact Imitation

Input: $\mathcal{T}$ = Training queries, and τ = search bound

```
Initialize the set of ranking examples D = ∅
for each training query Q ∈ T do
    Compute the target query plan (s_0^*, a_0^*), ··· , (s_τ^*, ∅)
    for each search step t = 0 to τ − 1 do
        Generate ranking examples R_t to imitate (s_t^*, a_t^*)
        Aggregate training data: D = D ∪ R_t
    end for
end for
Π_EI = Rank-Learner(D)
return greedy policy Π_select
```

6. Experiments

In this section, we first describe our experimental setup. Subsequently, we present the empirical evaluation and analysis of our SCALE framework.

6.1. Experimental Setup

Datasets. We employ three real-world open knowledge graphs: 1) YAGO[1] contains 2.6M entities (e.g., people, companies, cities), 5.6M relationships, and 10.7M nodes and edge labels, extracted from several knowledge bases including Wikipedia; 2) DBpedia[2] consists of 3.9M entities, 16.8M edges, and 14.9M labels; and 3) Freebase[3], a more diversified knowledge graph that contains 40.3M entities, 180M edges, and 81.6M labels.

Query Workload. We developed a query generator to produce both training and testing queries following the DBPSB benchmark (32). First, we generate a set of query templates, each has a topology sampled from a graph category, and is assigned with labels (specified as "entity types") drawn from top 20% most frequent labels in the real-world graphs. We created 20 templates to cover common entity types, and generated a total of $2K$ queries by instantiating the templates. The maximum size of the generated query graph (total number of nodes and edges) was 13. We employed 50% queries for training, 20% for validation, and 30% for testing respectively.

ORACLE **Policy Implementation.** We performed heuristic guided best-first beam search with different beam widths $b = \{1, 5, 10, 20, 50, 100\}$ and search bound τ values to compute high-quality target selection plans for each training query. We select the oracle policy (combination of b and τ values) based on both computation time and error.

SCALE **Implementation.** We employed `SVM-Rank` (25) as our base rank learner. The hyper-parameter C to trade-off training error and generalization error was tuned based on the validation data. SCALE(EI) and SCALE(DAGGER) corresponds to policy learning via exact imitation and DAgger algorithms respectively. We performed 5 iterations of DAgger with $\beta = 0.8$ and exponential decay. We selected the final policy based on the performance on validation data.

STAR Implementation. We implemented the state-of-the-art graph query processing approach STAR (42) in Java using the Neo4j (`neo4j.com`) graph database system.

Evaluation Metrics. We evaluate STAR, ORACLE, and SCALE using two metrics: 1) *Speed.* For a given graph query Q and data graph G, the response time of an algorithm $\mathcal{P}$, denoted as $\texttt{Time}(\mathcal{P}, Q)$, refers to the total CPU time taken from receiving the query to finding the answer; and 2) *Error.* We employ Levenshtein distance function (34) to compute node/edge label similarity, and R-WAG's ranking function(38) to compute the matching score $F(\cdot)$. Suppose an algorithm $\mathcal{P}$ produces answer $\hat{A}$ for a given query Q. We compute the error, $\texttt{Error}(\mathcal{P}, Q)$ as $1 - \frac{F(Q,\hat{A})}{\texttt{SMAX(Q)}}$, where `SMAX(Q)` is the maximum possible score according to $F(.)$ for a candidate answer of query Q. `Error` metric directly depends on the evaluation algorithm employed to compute the answer $\hat{A}$. `Error(Dual-Sim)` and `Error(VF2)` refer to the error metrics using dual simulation and VF2 algorithms respectively. Note that, dual simulation is *optimistic* in terms of evaluation and cheap in terms of computation; and VF2 is *exact* and computationally very expensive.

We conducted all our experiments on a Linux machine with 2.4 GHz CPU and 256GB RAM configuration. All experiments are conducted 3 times and averaged results are presented. We report the average metrics (computation time and error metric) over all testing queries.

1. mpi-inf.mpg.de/yago
2. dbpedia.org
3. freebase.com

6.2. Results and Analysis

We organize our results along different dimensions.

ORACLE **Policy Performance.** The performance of oracle depends on both beam size b and search bound τ. We employed `Error(Dual-Sim)` to calibrate the performance of oracle. We didn't see any noticeable improvements in the error beyond τ=100. Fig 3 shows the `Error(Dual-Sim)` of oracle with different values of τ and b for all the three knowledge graphs. We make the following observations: 1) We are able to uncover very high-quality subgraphs with very small τ and b values. This shows the effectiveness of our oracle construction procedure; 2) The error decreases as τ increases. However, the improvements are very small beyond τ=25 for all beam sizes; 3) The error decreases as b increases, but the improvements are very small beyond b=10 and they saturate at b=100. Since imitation learning algorithms require several calls to the oracle policy, policy learning won't scale for very large τ and b values. Therefore, we employ b=10 and τ=25 as our oracle policy to train the selection policy, which is a reasonable choice based on the above observations.

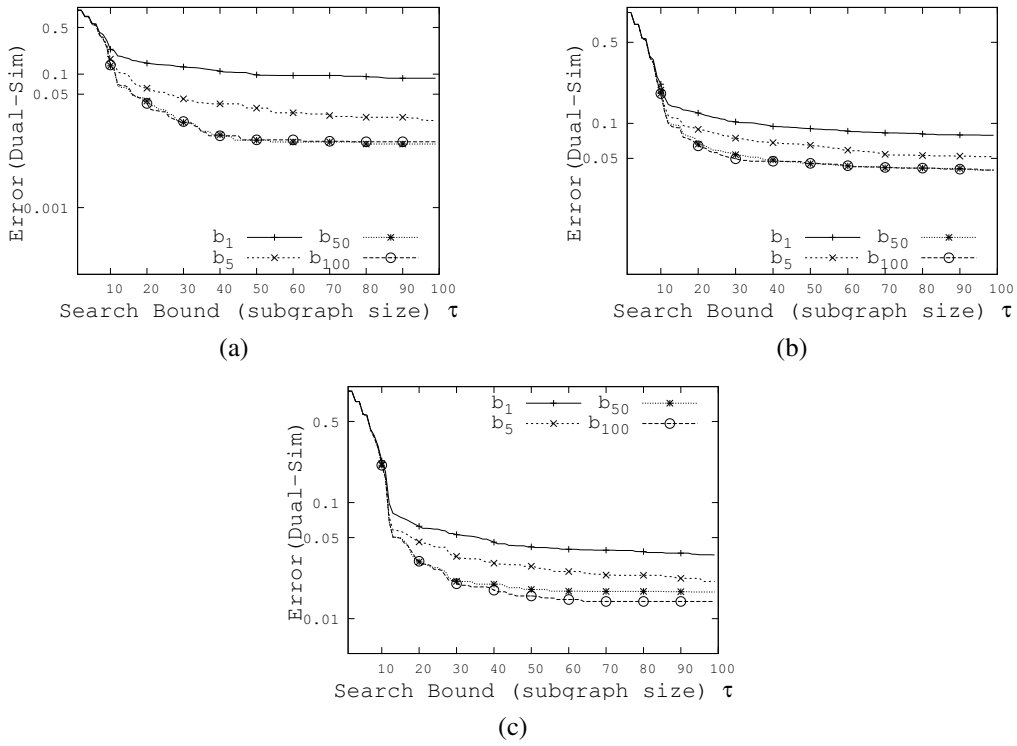


Figure 3: ORACLE policy performance with search bound τ value and different beam sizes: b_1, b_5, b_{50}, and b_{100} stands for beam size of 1, 5, 50, and 100 respectively. We show the `Error(Dual-Sim)` metric in log-scale: (a) YAGO, (b) Freebase, and (c) DBpedia

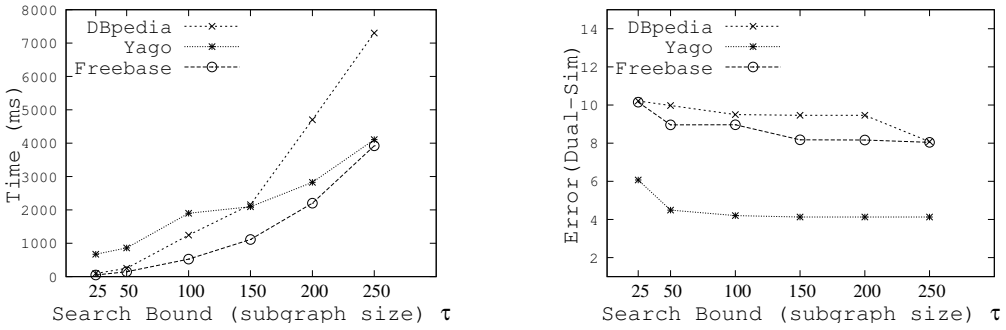
SCALE **vs.** ORACLE**.** ORACLE policy is computationally expensive due to the large beam size. To improve the computational efficiency, SCALE learns greedy policies. We compare SCALE with the oracle policy to understand how well the learner is able to mimic the search behavior of oracle. Table 1 shows the results of ORACLE (b=10), ORACLE (b=100), SCALE(EI), and SCALE(DAGGER) with τ=100. We can see that the error of SCALE(EI) in general is very close to the oracle policy

across all the datasets. SCALE(EI) loses at most 3% in error and significantly improves speed when compared to ORACLE (b=10) that was used for training. For example, SCALE(EI) is 12 times faster than ORACLE (b=10) with less than 1% loss in error for Freebase. Interestingly, the search time for SCALE is lowest for Freebase even though it is the largest graph. The reason is that the average number of actions available to grow the subgraph is smaller in Freebase when compared to the other graphs. This is a very advantageous property of SCALE. We saw very small improvement in error when the selection policy is trained with DAgger when compared to training with exact-imitation. However, training with DAgger is computationally excessive as learner aggressively queries the ORACLE policy for supervision. From now on, we will present all the results of SCALE with the policy trained via DAgger.

	YAGO		Freebase		DBpedia	
	Error	Time(s)	Error	Time(s)	Error	Time(s)
STAR	4.56%	5.81	11.58%	19.98	7.79%	9.69
ORACLE ($b = 10$)	2.80%	8.15	7.74%	6.54	6.82%	19.14
ORACLE ($b = 100$)	1.81%	14.61	6.56%	32.06	6.42%	25.98
SCALE(EI)	4.32%	2.05	8.45%	0.68	9.61%	1.47
SCALE(DAGGER)	4.13%	1.90	8.38%	0.53	9.46%	1.24

Table 1: SCALE vs. ORACLE and STAR with τ=100: Time in seconds and `Error(VF2)`.

SCALE **Performance vs. Search Bound τ.** The performance of SCALE in terms of computation time and error metric directly depend on the search bound τ. Figure 4 shows the performance of SCALE as a function of τ for all the three knowledge graphs. We can see that the computation time grows as τ increases. The growth is significant for DBpedia, which is due to the large number of candidate matches. On the other hand, the error decreases monotonically as τ increases. However, the improvement in error is very small to none beyond τ=50.



(a) (b)

Figure 4: SCALE performance vs. search bound τ: (a) Time in milli seconds (b) `Error(Dual-Sim)`.

SCALE **Performance vs. Beam Size b.** SCALE uses greedy search guided by the policy to perform selection. It is possible to improve the error of SCALE by adding some more search: performing beam search ($b > 1$) by plugging the learned greedy heuristic function. Table 2 shows the error and

computation time of SCALE ($b > 1$) for all the three knowledge graphs. SCALE ($b > 1$) improves the error metric ranging from 1 to 3% at the expense of increased computation time.

	YAGO		Freebase		DBpedia	
	Error	Time(s)	Error	Time(s)	Error	Time(s)
SCALE (b=2)	3.09%	3.16	8.37%	1.72	8.73%	3.50
SCALE (b=3)	2.70%	8.08	6.90%	3.18	8.29%	6.86
SCALE (b=4)	1.92%	12.42	6.81%	6.01	8.10%	11.23
SCALE (b=5)	1.37%	12.47	5.39%	8.56	7.65%	11.58
SCALE (b=10)	1.00%	17.29	5.54%	17.56	6.71%	16.69

Table 2: SCALE performance w/ τ=100 vs. beam size b: Time in seconds and `Error(VF2)`.

SCALE vs. STAR. ORACLE (b=10) and ORACLE (b=100) perform better than STAR in terms of error metric, but they take more time than STAR (except for Freebase). The error metric of SCALE is better than STAR (except DBPedia) and is faster by 3 to 38 times when compared STAR. For example, SCALE is 38 times faster and improves the error by 3.2% over STAR on Freebase knowledge graph. Indeed, STAR is very slow on dense graphs such as Freebase, where we expect more candidate matches for each star-query. We also investigated the reasons for the anomalous behavior of SCALE on DBpedia. We found that label distribution is the main reason. Specifically, nodes with some labels have a very high degree and results in large branching factor during search. As a result, the corresponding learning problem becomes very hard due to the large number of ranking constraints. SCALE (b=2) further improves the error and is still faster than STAR.

In summary, our SCALE framework can trade-off speed and accuracy of graph search by varying search bound τ and beam size b. Experimental results show that SCALE is able to produce highly accurate answers with only small graphs of size $\approx$100, and achieved significant speedup over a state-of-the-art graph search algorithm (STAR) with better or small loss in accuracy. We also performed experiments for search bound $\tau > 100$, but did not see noticeable improvements.

Training and Testing on Different Knowledge Graphs. The learned selection policy can be seen as a function that maps search states to appropriate actions via features. Since the feature definitions are general, and include different structural and syntactical properties of the data graph and query graph, one could hypothesize that the learned knowledge is general and can be used to search different data graphs. To test this hypothesis, we learned selection policies on each of the three knowledge graphs; and tested each policy on individual knowledge graphs. We make the following observations from Table 3. The learned selection policies generalize very well to datasets that are not used for their training. We get the best performance when training and testing are done on the same dataset.

Training and Testing on Different Query Templates. We tested the learned selection policy on query templates that are not used to generate the training queries to evaluate its generalization behavior. Table 4 shows the results with testing queries from old and new query templates. We can see that the selection policy generalizes reasonably well to new query templates and the loss in performance is very small.

In general, we need to update the policy whenever the distribution of queries and/or data graph changes significantly.

		Testing Graph		
		YAGO	Freebase	DBpedia
Training graph	YAGO	**4.13%**	4.53%	4.20%
	Freebase	10.78%	**8.38%**	10.73%
	DBpedia	9.72%	9.71%	**9.46%**

Table 3: `Error(VF2)` of SCALE with training and testing on different knowledge graphs.

	New Template	Old Template
YAGO	4.43%	4.10%
DBpedia	8.80%	8.38%
Freebase	10.94%	9.46%

Table 4: `Error(VF2)` of SCALE with training and testing on different query templates.

6.3. Limitations and Potential Solutions

In this section, we discuss the limitations of our approach along with ways to overcome them.

1) Non-Localized Graph Queries. Our SCALE approach can currently handle localized graph queries: queries whose matches exist in a local neighborhood of the data graph. It is an interesting open problem to extend SCALE to handle non-localized graph queries (e.g., simulation and reachability queries).

2) Scalability of Learning Approach. We can employ advanced imitation learning approaches such as LOLS (1) to potentially improve over the reference (oracle) policy. However, this approach is relatively more expensive than DAgger. We need to explore active learning approaches to selectively query the oracle policy to improve the efficiency of learning.

3) Selection Policy Representation. We learned linear ranking functions from hand-designed features Ψ over state-action pairs. To improve the performance, we can leverage the recent advances in deep neural networks for learning good representations for the policy (6).

7. Related Work

Our work is related to a sub-area of AI called speedup learning (17). Specifically, it is an instance of inter-problem speedup learning. Reinforcement learning (RL), imitation learning (IL), and hybrid approaches combining RL and IL have been explored to learn search control knowledge from training problems in the context of diverse application domains. Some examples include job shop scheduling (44), deterministic and stochastic planning (41; 35), natural language processing (24; 21; 22; 30), hardware design optimization (7; 8; 9), and mixed-integer programming solvers (23; 26). Our work explores this speedup learning problem for graph search, a novel problem with numerous applications. We developed a novel learning framework called SCALE, and formalized and solved the corresponding speedup learning problem using imitation learning.

There is some work on loose integration of supervised learning techniques in the context of query optimization (20; 18; 19). In a concurrent work (33), Namaki and colleagues developed a learning to plan (L2P) framework for TA-style algorithms that is similar in spirit to our work. However, SCALE is more general than L2P framework as it is not tied to a specific query reasoner and query class. The resource-bounded (RB) querying framework (16) is closely related to our work, but SCALE is more general than RB approach for the following reasons: 1) RB approach assumes a known personalized node as seed node, which results in a easier search problem. SCALE does not make such assumptions; 2) RB approach works for only complete matches and the concept of best answer is not applicable. SCALE can work with both partial and complete matches; and 3) The search control knowledge is specified by the human designer in RB approach, whereas it is automatically learned in SCALE.

8. Summary and Future Work

We developed a novel learning framework called SCALE to improve the computational-efficiency of searching large-scale knowledge graphs. Our framework allows to automatically trade-off speed and accuracy of graph search as per the needs of the application. The key idea is to select a small part of the data graph that is sufficient to answer a given query in order to satisfy the specified constraints on speed/accuracy. We integrated learning and search to quickly select high-quality subgraphs in a data-driven manner via training queries. We showed that our SCALE framework was able to produce highly accurate answers with only small graphs of size $\approx$100 and achieved significant speedup over a state-of-the-art graph search algorithm (STAR) with better or small loss in accuracy across multiple knowledge graphs.

Future work includes exploring deep learning based policies to match the accuracy of the oracle policy; scaling up the learning framework to handle large number of batch/streaming queries by exploring active learning techniques; and deploying the learning system for real-world applications.

Acknowledgements. This work was supported in part by the NSF grant #1543656 and the ARO grant W911NF-17-1-0485 from their computing sciences program.

References

[1] Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. Learning to search better than your teacher. In *ICML*, 2015.

[2] Jiefeng Cheng, Xianggang Zeng, and Jeffrey Xu Yu. Top-k graph pattern matching over large graphs. In *ICDE*, 2013.

[3] George Chin Jr, Sutanay Choudhury, John Feo, and Lawrence Holder. Predicting and detecting emerging cyberattack patterns using streamworks. In *Proceedings of the 9th Annual CISRC*, pages 93–96, 2014.

[4] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. An improved algorithm for matching large graphs. In *International Workshop on Graph-based Representations in Pattern Recognition*, pages 149–159, 2001.

[5] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, 2004.

[6] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *ICML*, pages 2702–2711, 2016.

[7] Sourav Das, Janardhan Rao Doppa, Daehyun Kim, Partha Pratim Pande, and Krishnendu Chakrabarty. Optimizing 3D NoC design for energy efficiency: A machine learning approach. In *ICCAD*, pages 705–712, 2015.

[8] Sourav Das, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. Design-space exploration and optimization of an energy-efficient and reliable 3D small-world network-on-chip. *TCAD*, 2016.

[9] Sourav Das, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. Monolithic 3D-enabled high-performance and energy efficient network-on-chip. In *ICCD*, 2017.

[10] Xiaofeng Ding, Jianhong Jia, Jiuyong Li, Jixue Liu, and Hai Jin. Top-k similarity matching in large graphs with attributes. In *DASFAA*, 2014.

[11] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. HC-Search: A learning framework for search-based structured prediction. *JAIR*, 50:369–407, 2014.

[12] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Structured prediction via output space search. *JMLR*, 15: 1317–1350, 2014.

[13] Janardhan Rao Doppa, Jun Yu, Chao Ma, Alan Fern, and Prasad Tadepalli. HC-Search for multi-label prediction: An empirical study. In *AAAI*, 2014.

[14] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003.

[15] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu, and Yunpeng Wu. Graph pattern matching: From intractable to polynomial time. *PVLDB*, 3(1):264–275, 2010.

[16] Wenfei Fan, Xin Wang, and Yinghui Wu. Querying big graphs within bounded resources. In *SIGMOD*, 2014.

[17] Alan Fern. Speedup learning. In *Encyclopedia of Machine Learning*. 2010.

[18] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet L Wiener, Armando Fox, Michael Jordan, and David Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *ICDE*, 2009.

[19] Chetan Gupta, Abhay Mehta, and Umeshwar Dayal. Pqr: Predicting query execution times for autonomous workload management. In *ICAC*, 2008.

[20] Ragib Hasan and Fabien Gandon. A machine learning approach to sparql query performance prediction. In *IEEE/WIC/ACM*, 2014.

[21] He He, Hal Daumé III, and Jason Eisner. Imitation learning by coaching. In *NIPS*, 2012.

[22] He He, Hal Daumé III, and Jason Eisner. Dynamic feature selection for dependency parsing. In *EMNLP*, 2013.

[23] He He, Hal Daumé III, and Jason Eisner. Learning to search in branch and bound algorithms. In *NIPS*, 2014.

[24] Jiarong Jiang, Adam R. Teichert, Hal Daumé III, and Jason Eisner. Learned prioritization for trading off accuracy and speed. In *NIPS*, 2012.

[25] Thorsten Joachims. Training linear SVMs in linear time. In *KDD*, 2006.

[26] Elias Boutros Khalil, Pierre Le Bodic, Le Song, George L. Nemhauser, and Bistra N. Dilkina. Learning to branch in mixed integer programming. In *AAAI*, 2016.

[27] Jinha Kim, Hyungyu Shin, Wook-Shin Han, Sungpack Hong, and Hassan Chafi. Taming subgraph isomorphism for rdf query processing. *VLDB*, pages 1238–1249, 2015.

[28] Michael Lam, Janardhan Rao Doppa, Sinisa Todorovic, and Tom Dietterich. HC-Search for structured prediction in computer vision. In *CVPR*, 2015.

[29] Jiaheng Lu, Chunbin Lin, Wei Wang, Chen Li, and Haiyong Wang. String similarity measures and joins with synonyms. In *SIGMOD*, 2013.

[30] Chao Ma, Janardhan Rao Doppa, John Walker Orr, Prashanth Mannem, Xiaoli Z. Fern, Thomas G. Dietterich, and Prasad Tadepalli. Prune-and-Score: Learning for greedy coreference resolution. In *EMNLP*, 2014.

[31] Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai, and Tianyu Wo. Capturing topology in graph pattern matching. *PVLDB*, 5(4):310–321, 2011.

[32] Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. Dbpedia sparql benchmark–performance assessment with real queries on real data. *The Semantic Web–ISWC 2011*, pages 454–469, 2011.

[33] Mohammad Hossein Namaki, F. A. Rezaur Rahman Chowdhury, Md Rakibul Islam, Janardhan Rao Doppa, and Yinghui Wu. Learning to speed up query planning in graph databases. In *ICAPS*, pages 443–451, 2017.

[34] Gonzalo Navarro. A guided tour to approximate string matching. *CSUR*, 33(1):31–88, 2001.

[35] Jervis Pinto and Alan Fern. Learning partial policies to speedup MDP tree search. In *UAI*, 2014.

[36] Eric PrudHommeaux, Andy Seaborne, et al. Sparql query language for rdf. *W3C recommendation*, 15, 2008.

[37] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[38] S. B. Roy, T. Eliassi-Rad, and S. Papadimitriou. Fast best-effort search on graphs with multiple attributes. *TKDE*, 27(3):755–768, 2015.

[39] Martin Theobald, Holger Bast, Debapriyo Majumdar, Ralf Schenkel, and Gerhard Weikum. Topx: efficient and versatile top-k query processing for semistructured data. *VLDB*, 2008.

[40] Andreas Wagner, Veli Bicer, and Thanh Tran. Pay-as-you-go approximate join top-k processing for the web of data. In *ESWC*. 2014.

[41] Yuehua Xu, Alan Fern, and Sung Wook Yoon. Learning linear ranking functions for beam search with application to planning. *JMLR*, 10:1571–1610, 2009.

[42] Shengqi Yang, Fangqiu Han, Yinghui Wu, and Xifeng Yan. Fast top-k search in knowledge graphs. In *ICDE*, 2016.

[43] X. Zeng, J. Cheng, J. Yu, and S. Feng. Top-k graph pattern matching: A twig query approach. *WAIM*, 2012.

[44] Wei Zhang and Thomas G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, 1995.

[45] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. Natural language question answering over RDF: a graph data driven approach. In *SIGMOD*, 2014.