Maverick: Discovering Exceptional Facts from Knowledge Graphs

ABSTRACT

We present Maverick, a general, extensible framework that discovers exceptional facts about entities in knowledge graphs. To the best of our knowledge, there was no previous study of the problem in the domain of knowledge graphs. We model an exceptional fact about an entity of interest as a context-subspace pair, in which a subspace is a set of attributes and a context is defined by a graph query pattern of which the matches include the entity. The entity is exceptional among the entities in the context, with regard to the subspace. The search spaces of both patterns and subspaces are exponentially large. Maverick conducts beam search on a Hasse diagram of patterns. The beam search uses a match-based pattern construction method which evades the evaluation of invalid patterns. Furthermore, it exploits two pruning rules to exclude irrelevant patterns and to avoid repeated constructions of patterns. It also applies two heuristics to select promising patterns to form the beam in each iteration. For efficiently finding highly-scored subspaces in each context, Maverick uses a set enumeration tree to traverse the subspaces and exploits the upper bound properties of exceptionality scoring functions to prune the set enumeration tree. Experiment results using real-world datasets demonstrated substantial performance improvement of the proposed framework over the baselines as well as its effectiveness in discovering exceptional facts.

1 INTRODUCTION

Knowledge graphs such as DBpedia [4], Freebase [6], Wikidata [28], and YAGO [23] record properties of and relationships between real-world entities. These data are used in numerous applications, including search, recommendation, and business intelligence. This paper introduces Maverick, a framework that, given an entity in a knowledge graph, discovers *exceptional facts* about the entity. Informally, such exceptional facts separate the entity from many other entities. Consider several factual statements in published news articles:

(1) "Denzel Washington followed Sidney Poitier as only the second black to win the Best Actor award." (*abcnews.go.com*)
(2) "This was Brazil's first own goal in World Cup history ..." (*yahoo.com*)
(3) "Hillary Clinton becomes first female presidential nominee." (*chicagotribune.com*)

An exceptional fact consists of three components: an *entity of interest*, a *context*, and a set of *qualifying attributes*. In each exceptional fact, among all entities in the context, the entity of interest is one of

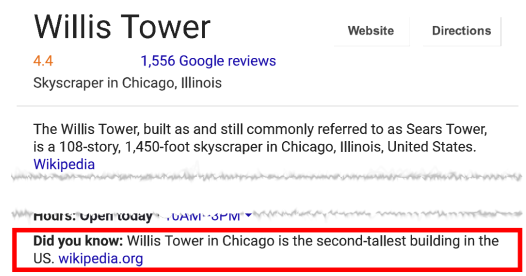


Figure 1: Google's "Did you know" feature displays exceptional facts about entities in search results. (*Google.com*)

the few or even the only one that bears a particular value combination on the qualifying attributes. For example, in the above statement 1, the entity of interest is Denzel Washington, the context is the Academy Award Best Actor winners, and the qualifying attribute is ethnicity.

Discovery of exceptional facts is useful to important applications such as computational journalism [8, 9], recommendation systems, and data cleaning. a) In *fact-finding* [14, 15, 24, 31, 34], journalists are interested in monitoring data and discovering attention-seizing factual statements such as the aforementioned examples. These facts help make news stories substantiated and interesting, and they may even become leads to news stories. b) In *fact-checking* [13, 32], for vetting the statements made by humans, fact-checkers at news organizations such as The Washington Post, CNN, and PolitiFact can compare the statements with automatically-discovered facts. For example, an algorithm may find that Hillary Clinton is the second female presidential nominee, which contradicts with the statement 3 above. [1] c) Exceptional facts can help promote friends, news, products, and search results in various recommendation systems. For example, Google's "Did you know" feature displays exceptional facts about entities in search results, as shown in Fig. 1. d) When the discovered facts are inconsistent with known truth or apparent common sense, it reveals incomplete data or data errors. Such insights aid knowledge base cleaning and completion. For example, the above statement 3 may be generated using an incomplete source that misses the nomination of Victoria Woodhull.

Given an entity in a knowledge graph, an integer k, and an exceptionality scoring function, the objective of *exceptional fact discovery* is to find the top-k highest scored pairs of (context, attribute set). The entity is exceptional with regard to the attributes, while at the same time belonging to the context together with other entities. This description hinges upon two concepts—context and attribute—which we explain below.

- The *attributes* of an entity are the entity's incoming/outgoing edge labels, and the attribute values are the entity's direct neighbors. For example, Fig. 2 is an excerpt of a knowledge graph about FIFA World Cup, in which the edge labeled *awarded-to* from node

[1] The first female presidential nominee was Victoria Woodhull, according to *http://www.snopes.com/victoria-woodhull-hillary-clinton/*.

SIGMOD'18, June 2018, Houston, Texas USA

ACM ISBN 123-4567-24-567/08/06...$15.00
https://doi.org/10.475/123_4

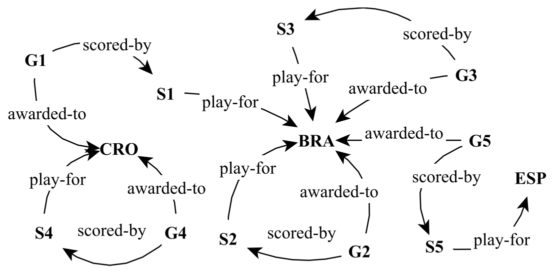


Figure 2: An excerpt of a knowledge graph.

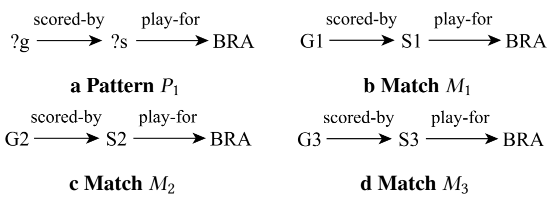


Figure 3: Pattern P_1 and variable ?g define a context consisting of all the goals scored by BRA players; M_1, M_2, M_3 are matches to P_1 in Fig. 2.

G1 to CRO captures the fact that the goal is awarded to the team Croatia. Entity G1 has two attributes *scored-by* and *awarded-to*, with values S1 and CRO, respectively.

- A *context* is a set of entities sharing some common characteristics defined in a pattern query. In Fig. 3a, pattern P_1 and the variable ?g in it define a context C_1 of all the goals scored by players of team Brazil. Figs. 3b-3d show P_1's matches in Fig. 2. For instance, match M_1 (Fig. 3b) is a subgraph of Fig. 2, in which ?g of P_1 is mapped to G1. Hence, G1 belongs to context C_1. Similarly, G2 and G3 in Fig. 2 also belong to C_1 based on M_2 and M_3, while G4 and G5 are not part of C_1.
- With respect to a *subspace* (i.e., a set of attributes), an entity is exceptional in a context if its attribute values deviate from the values of other entities in the same context. For example, the value of attribute *awarded-to* for G1 is CRO, while the value is BRA for both G2 and G3. The degree of exceptionality of an entity varies by different contexts and subspaces. For instance, one interpretation of statement 1 is that the context is the Academy Award Best Actor winners and the qualifying attribute is ethnicity; an alternative interpretation is that the context is all African Americans and the qualifying attribute is the award. Under some definitions of exceptionality, the second interpretation may render Denzel Washington more exceptional, since there are a lot more African Americans than winners of the award.

A holistic solution to exceptional fact discovery may be expected to synthesize whatever types of available data (structured databases, graphs, text, and so on), which is beyond the scope of this paper. Instead, our focus is on knowledge graphs which are becoming increasingly important to analytics and intelligence applications. To the best of our knowledge, there is no previous study on discovering exceptional facts about entities in knowledge graphs. The two most related areas are *outlier detection* in graphs [11, 19, 25, 29] and *outlying aspect mining* [2, 3, 10, 14, 24, 27, 30]. Duan et al. [10] and Vinh et al. [27] discussed the differences between these two areas. They achieve different goals. Outlier detection searches for all outlying objects among a set of objects. Outlying aspect mining, however, focuses on only one given object and returns the subspaces of attributes in which the object is relatively outlying, regardless of its true degree of outlyingness. In terms of objectives and problem modeling, the exceptional fact discovery problem formulated in this paper is closer to outlying aspect mining than outlier detection. However, it focuses on graph data. In contrast, existing outlying aspect mining methods [3, 14, 24, 30] assume a single relational table. These methods take a tuple as input and returns two disjoint attribute sets. The first set of attributes define the context, i.e., the tuples having values identical to that of the input tuple on the attributes. On the second set of attributes, the input tuple has peculiar values compared to other tuples belonging to the context.

However, these methods for outlying aspect mining cannot be effectively applied to knowledge graphs, since they are specifically devised for single tables only. A seemingly plausible idea can be to represent a knowledge graph as a single table and then to apply the existing methods on the table. Consider the single-table model of RDF proposed in [7]. When adapting it for a knowledge graph, each tuple (row) is for an entity v and each attribute (column) corresponds to an edge label in the knowledge graph. The attribute is also associated with an edge direction—either incoming into or outgoing from v. The value at the junction of the row and the column is an entity or a set of entities adjacent to v via edges with the label and direction given by the column. Given this single-table representation of the knowledge graph, at least a few major problems render the existing outlying aspect mining methods inapplicable. *First*, in these methods a context, defined by a set of attributes, consists of the tuples having values identical to that of the input tuple. In other words, the context is the result of a conjunctive query over the attributes. For knowledge graphs, however, a context is defined by a graph pattern query, which cannot be captured by conjunctive queries on attributes in the aforementioned single-table representation. More specifically, an edge in the pattern may not be adjacent to the input entity and thus does not correspond to any of the entity's attributes. Hence, evaluating a pattern may involve self-joins of the single-table. Existing outlying aspect mining methods are not designed to accommodate joins. *Second*, the aforementioned set values in the single-table representation are not considered in the existing methods. An adaptation of the methods will thus require at least joins which, as mentioned above, are not supported by the methods. *Third*, due to the heterogeneity and scale of a large knowledge graph, such a single-table is extremely wide and sparse, which is well beyond the capacity of the existing methods because of the intrinsic exponential complexity of the problem's search space.

To discover the exceptional facts about an entity, we must explore two extremely large search spaces, one of patterns and the other of attribute subspaces. (Section 5.1 shows that the number of patterns is at least exponential in the size of the graph.) It is also clear that the number of subspaces is exponential in the number of attributes since a subspace is a combination of attributes. It is not computationally feasible to exhaustively enumerate all possible patterns and subspaces. Furthermore, it is challenging to prune patterns and subspaces, due to the non-existence of *downward closure property* (i.e., *anti-monotone property*) on typical exceptionality scoring functions.

To tackle these challenges, this paper introduces Maverick, a beam-search based framework. Given an input entity, Maverick discovers the top-k context-subspace pairs that give the entity the highest expectionality scores. Maverick allows an application to plug in any exceptionality scoring function based on the application needs. Conceptually, Maverick organizes the search space of patterns as a partial order defined by the subsumption relation on patterns and the search space of attribute subspaces as a set enumeration tree [22]. Intuitively, the search for top-k context-subspace pairs is performed in a nested-loop fashion in which the outer loop enumerates patterns and the inner loop enumerates subspaces. Maverick conducts breath-first beam search [33] on the space of patterns, starting from a pattern with a single variable node. On each visited pattern, Maverick applies a set of rules and heuristics to prune its children so that Maverick visits at most w patterns at each level, where w is the beam width. Each visited pattern is evaluated over the knowledge graph to obtain the contexts it defines. For each context, Maverick calculates the input entity's exceptionality scores in different subspaces. It exploits an upper bound for exceptionality score to guide the traversal of the subspaces. The supersets of a subspace are pruned if their upper-bound scores are below the current top-k scores.

The paper reports the results of experiments on two real-world knowledge graphs, which verify Maverick's effectiveness in finding exceptional facts. The experiments compared the performance of a breath-first search method and the beam search method coupled with different candidate-selection heuristics. The experiment results establish that, even though the breath-first search method may evaluate more patterns in a fixed time frame than the beam search methods, it is not as effective as the beam search method using the proposed heuristics. The results also show that the pruning rules can significantly reduce the number of candidates and thus improve the efficiency. We have also included some exceptional facts discovered by Maverick to demonstrate its practicality.

To summarize, this paper makes the following contributions:

- It motivates and formulates the novel problem of exceptional fact discovery in knowledge graphs. Although a similar problem has been studied on single-table data model, there is no such prior study on graphs. The approaches developed for single-table data model are inapplicable on graph data.
- It tackles the non-trivial challenges in modeling the problem as finding top-k context-subspace pairs. It presents a general, extensible solution framework Maverick that allows for customization.
- In Maverick, several non-trivial components work in tandem. They ensure the efficacy of Maverick by beam search on patterns, match-based pattern construction, pattern pruning rules and selection heuristics, using set numeration trees to traverse the subspaces for each context, and pruning set enumeration trees using the upper bound properties of exceptionality scoring functions.
- Experiment results using real-world data demonstrate Maverick's efficiency and effectiveness in discovering exceptional facts.

2 PROBLEM FORMULATION

In this section we formally define the data model of knowledge graphs, the concepts of context, attribute, and subspace, and the problem of exceptional fact discovery.

Knowledge Graphs

A knowledge graph $G(V_G, E_G)$ is a labeled, directed multi-graph with node set V_G and edge set E_G. Each node $v \in V_G$ represents an entity and has a unique identifier $id(v)$. (Without loss of generality, we use an entity's name as its identifier in the ensuing examples, assuming entity names are unique.) An edge $e{=}(v_i, l, v_j) \in E_G$ denotes a directed relationship from entity v_i to entity v_j. It has a label $l \in L$, where L is the universal set of edge labels. Multiple edges outgoing from (or incoming into) v can have the same label.

In Fig. 2, there are three kinds of entities: goals (e.g., G1), players (e.g., S1), and teams (e.g., BRA). Three different types of edge labels represent different relationships: each player plays for a team (*play-for*), and each goal is scored by a player (*scored-by*) and is awarded to a team (*awarded-to*). For example, there is an own goal, as G1 is scored by S1, a player of BRA, but awarded to CRO.

Patterns and Contexts

Definition 2.1 (Pattern P). A pattern is a weakly connected graph[2] $P(V_P, E_P)$. It has two kinds of nodes, which are entities (Y_P) and variables (X_P). More formally, $V_P = Y_P \cup X_P$, where $Y_P \subseteq V_G$ and $Y_P \cap X_P = \emptyset$. △

Definition 2.2 (Match M). A match $M(V_M, E_M)$ to a pattern $P(V_P, E_P)$ is a subgraph of G ($V_M \subseteq V_G$ and $E_M \subseteq E_G$) such that there exists a bijection $f : V_P \rightarrow V_M$ satisfying the following conditions:

- $|V_M| = |V_P|$, $|E_M| = |E_P|$;
- $\forall\ (v_i, l, v_j) \in E_P \Rightarrow (f(v_i), l, f(v_j)) \in E_M$;
- $\forall\ (u_i, l, u_j) \in E_M \Rightarrow (f^{-1}(u_i), l, f^{-1}(u_j)) \in E_P$;
- $\forall\ v \in Y_P \Rightarrow id(f(v)) = id(v)$.

In short, a subgraph M of G is a match to pattern P if M is edge-isomorphic to P and, for each non-variable node v in P, $f(v)$ has the same identifier. △

Definition 2.3 (Range of Variable R_x^P). Let $\mathcal{M}_P$ be all the matches to pattern P in a knowledge graph G. For a variable $x \in X_P$, the range of x, denoted R_x^P, is a set of entities defined as

$$R_x^P = \{f(x) \mid M \in \mathcal{M}_P, f : V_P \rightarrow V_M\}. \quad △$$

For example, P_1 in Fig. 3a has two variable nodes, ?g and ?s. (To distinguish variables from entities, the names of variable nodes always start with the symbol ?.) Figs. 3b-3d show P_1's matches in Fig. 2. $R_{?g}^{P_1}$ = {G1, G2, G3} and $R_{?s}^{P_1}$ = {S1, S2, S3}.

Definition 2.4 (Context $C_v^{P,x}$). Given an entity v, a pattern P, a variable $x \in X_P$ such that $v \in R_x^P$, the context of v defined by P and x is denoted $C_v^{P,x}$ and $C_v^{P,x}{=}R_x^P$. △

For example, the context of G1 in the running example—goals scored by BRA players—is defined by pattern P_1 in Fig. 3a and variable $?g$ in the pattern: $C_{\text{G1}}^{P_1,?\text{g}} = R_{?\text{g}}^{P_1}$ = {G1, G2, G3}. On the other hand, since G1 $\notin R_{?\text{s}}^{P_1}$, ?s in P_1 does not define a context of G1. Note that a pattern may define multiple contexts of v, since v may be mapped to different variables in the pattern. For example, consider pattern P = {(?g, *awarded-to*, ?a), (?g, *scored-by*, ?s), (?s, *play-for*, ?t)}. It defines two different contexts of BRA: $C_{\text{BRA}}^{P,?\text{a}}$ = {CRO, BRA}, $C_{\text{BRA}}^{P,?\text{t}}$ = {ESP, BRA}.

Entity Attributes and Subspaces

Given an entity of interest v, an attribute corresponds to the label of an edge incoming into or outgoing from v, and its value is the entity at the other end of the edge. Note that we need to distinguish between incoming attributes and outgoing attributes since an entity can be both sources and destinations of edges of the same label. For instance, a person can have a manager and meanwhile be the manager of someone else.

[2] A weakly connected graph is a directed graph of which the corresponding undirected graph is connected.

Definition 2.5 (Entity Attributes A_v). Given an entity v, its attributes A_v is the union of its incoming and outgoing attributes: $A_v = A^{\mathrm{i}}_v \cup A^{\mathrm{o}}_v$. The incoming attributes are a set of edge labels $A^{\mathrm{i}}_v = \{(l, \leftarrow) \mid \exists(x, l, v) \in E_G\}$. Given an incoming attribute $a = (l, \leftarrow) \in A^{\mathrm{i}}_v$, v's value on attribute a is the set $v.a = \{x \mid (x, l, v) \in E_G\}$. Similarly, the outgoing attributes are $A^{\mathrm{o}}_v = \{(l, \rightarrow) \mid \exists(v, l, x) \in E_G\}$. Given an outgoing attribute $a = (l, \rightarrow) \in A^{\mathrm{o}}_v$, v's value is $v.a = \{x \mid (v, l, x) \in E_G\}$. △

Definition 2.6 (Subspace A). A subspace A is a subset of v's attributes, i.e., $A \subseteq A_v$. The projection of v's attribute values onto subspace A is denoted $v.A$, and $v.\varnothing = \texttt{null}$. △

For example, in Fig. 2, $A^{\mathrm{i}}_{\text{CRO}} = \{(\textit{play-for}, \leftarrow), (\textit{awarded-to}, \leftarrow)\}$; CRO.(*awarded-to*, $\leftarrow$) $= \langle\{$G1, G4$\}\rangle$ and $A^{\mathrm{o}}_{\text{G1}} = \{(\textit{scored-by}, \rightarrow), (\textit{awarded-to}, \rightarrow)\}$; G1.(*awarded-to*, $\rightarrow$) $= \langle\{$CRO$\}\rangle$. Let subspace $A = \{(\textit{scored-by}, \leftarrow), (\textit{play-for}, \rightarrow)\}$. $A_{\text{S1}} = A$ and S1.$A = \langle\{$G1$\}, \{$BRA$\}\rangle$.

Exceptionality Score

Definition 2.7 (Exceptionality Scoring Function χ). An exceptionality scoring function $\chi(v, A, C) \in \mathbb{R}$ measures entity v's degree of exceptionality with regard to subspace A in comparison with other entities in context C. △

Section 4.2 discusses several representative exceptionality scoring functions. Without loss of generality, we assume the range of χ is $[0, 1]$, with larger χ values implying greater exceptionality. We also set $\chi(v, A, C) = 0$ if $A \nsubseteq A_v$ or $v \notin C$, to make function χ total.

Definition 2.8 (Top-k Exceptional Facts F_v). With regard to an entity v, the *rank* of a context-subspace pair (C, A) is the number of context-subspace pairs with greater exceptionality scores, i.e., $rank(C, A) = |\ \{(C', A') \in \mathcal{C}_v \times A_v \mid \chi(v, A', C') > \chi(v, A, C)\}\ |$. $\mathcal{C}_v$ is the universe of v's contexts: $\mathcal{C}_v = \{C^{P,x}_v \mid P \in \mathcal{P}, x \in P, v \in R^P_x\}$, in which $\mathcal{P}$ is the universe of patterns over G, i.e., $\mathcal{P} = \{P(V_P, E_P) \mid V_P \subseteq X \cup V_G, E_P \subseteq (X \cup V_G) \times L \times (X \cup V_G), P(V_P, E_P)$ is weakly connected$\}$ where X is the universe of variables. (C, A) is a *top-k exceptional fact* if its rank is lower than k. Hence, the set of top-k exceptional facts about v, F_v, is defined as $F_v = \{(C, A) \in \mathcal{C}_v \times A_v \mid rank(C, A) < k\}$.[3] △

Problem Statement Given a knowledge graph G, an entity of interest v_0, an integer k, and an exceptionality scoring function χ, the problem of *exceptional fact discovery* is to find F_{v_0}—the top-k exceptional facts about v_0.

Continue the running example. With regard to entity G1, the context-subspace pair $(C^{P_1, ?g}_{\text{G1}}, \{(\textit{awarded-to}, \rightarrow)\})$ may be an exceptional fact. The context $C^{P_1, ?g}_{\text{G1}}$ is $\{$G1, G2, G3$\}$, i.e., the goals scored by BRA players. An interpretation of G1's exceptionality with regard to the pair is: among all the goals scored by BRA players, G1 is the only own goal.

3 OVERVIEW OF FRAMEWORK

We propose Maverick, an iterative framework for exceptional fact discovery. Intuitively, the process of discovering context-subspace pairs can be viewed as nested loops. The outer loop enumerates contexts, while the inner loop enumerates subspaces for each context. Given the entity of interest v_0, while subspace enumeration in the inner loop enumerates the subsets of A_{v_0}, the outer loop enumerates contexts by patterns, since each context of an entity is defined by a pattern and one of the pattern's variables (c.f. Definition 2.4). Conceptually, Maverick organizes all the possible contexts as a partial order on patterns, i.e., a Hasse diagram, in which each node is a pattern and each edge represents the subsumption (subgraph-supergraph) relationship between the two patterns. The essence of the outer loop is thus a traversal of the search space of patterns.

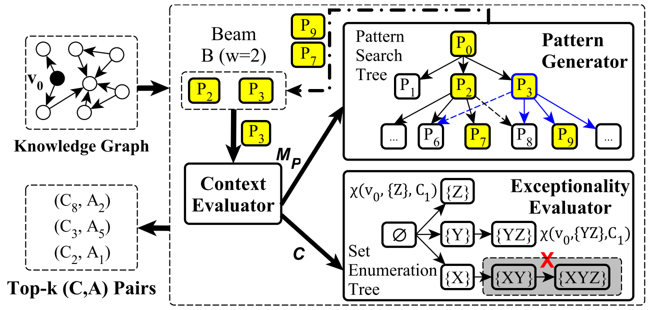


Figure 4: The framework of Maverick.

Algorithm 1: Discovering exceptional context-subspace pairs.

```
1  FACT-DISCOVER (G, v0, χ, k, w)
     Input: G : the knowledge graph; v0 ∈ V_G : the entity of interest;
            χ : the exceptionality scoring function; k : the size of
            output; w : the beam width
     Output: H : k most exceptional context-subspace pairs
2    P0 ← (V_P0 = {x0}, E_P0 = ∅) ;      // Initial state. x0 is a variable.
3    B ← {P0} ;                                                   // Beam.
4    i ← 1 ;                                           // Iteration number.
5    while B ≠ ∅ and i ≤ MAX_ITERATION do
6      i ← i + 1; B_tmp ← ∅;
7      foreach P ∈ B do
         // Obtain contexts of v0 and matches to P. (Section 3.1)
8        C^P_v0, M_P ← CONTEXT-EVALUATOR(P, v0, G);
9        foreach C ∈ C^P_v0 do
           // Exceptionality Evaluation. (Section 4)
10         𝒜 ← EXCEPTIONALITY-EVALUATOR(v0, C, k, χ);
11         foreach A ∈ 𝒜 do H ← H ∪ {(C, A)} ;
         // Find 𝒴 — the children of P. (Section 5)
12       𝒴 ← PATTERN-GENERATOR(v0, P, M_P, w, G);
13       B_tmp ← B_tmp ∪ 𝒴;
14     B ← top-w of B_tmp based on heuristics h ;          // Section 5.4
15   return top-k pairs in H based on exceptionality scores;
```

Given that the search space of patterns can be extremely large (Section 5), it is impractical to adopt breath-first, depth-first, or heuristic search approaches due to memory and time constraints [21]. To address this challenge, we propose to traverse the search space by *beam search* [5]. Since beam search maintains a "beam" of heuristically w best nodes and prunes all other nodes, it is not guaranteed to be complete or optimal. However, good solutions can be found quickly if the heuristic is sound enough.

Fig. 4 and Alg. 1 illustrate the framework of Maverick, which has three main components: Context Evaluator (CE), Exceptionality Evaluator (EE), and Pattern Generator (PG). The beam search at the outer loop starts with a pattern P_0 with a single variable node x_0 (Lines 2–3 in Alg. 1). The search results in a *pattern search tree*, of

[3]The size F_v may be greater than k due to ties in exceptionality scores and thus ranks.

Algorithm 2: Context evaluator.

1 **CONTEXT-EVALUATOR** $(P,\ v_0,\ G)$
2     $C^P_{v_0} \leftarrow \emptyset$; // The set of contexts defined by P.
3     $\mathcal{M}_P \leftarrow \texttt{match}(G, P)$; // Matches to P.
4   **foreach** $x \in X_P$ **do**
        // Refer to Definitions 2.3 and 2.4 for R^P_x and $C^{P,x}_{v_0}$.
5   **if** $v_0 \in R^P_x$ **then** $C^P_{v_0} \leftarrow C^P_{v_0} \cup \{C^{P,x}_{v_0}\}$;
6   **return** $(C^P_{v_0},\ \mathcal{M}_P)$;

which the root is P_0. At each iteration, Maverick maintains a beam B of a fixed size w (Lines 6, 13, 14). The beam consists of heuristically the best w patterns (e.g., P_2, P_3 in Fig. 4 where $w = 2$) at the visited level of the pattern search tree. For each pattern P in B, component CE obtains the matches $\mathcal{M}_P$ to the pattern and the corresponding contexts $C^P_{v_0}$ of v_0 (Line 8). For each context C in $C^P_{v_0}$ (e.g. C_1 in Fig. 4), component EE finds the top-k scored subspaces according to a given exceptionality scoring function χ (Line 10, and Section 4). Component PG finds the children of the visited pattern based on its matches (Line 12, and Section 5). Since there are usually much more children than what the beam size w allows, PG applies a set of rules (Section 5.3) and heuristics (Section 5.4) to prune the child patterns. Each child pattern is given a score that measures how promising it is according to the heuristics. The best w patterns among all the children of patterns in B will become the new beam B (Line 14), which is the input to the next iteration, e.g., $\{P_7, P_9\}$ in Fig. 4. The process ends when the limit on the number of iterations has reached. The limit is set to avoid overly-complex patterns which correspond to facts that are only convolutedly interesting. It also practically bounds the resource spent for running the algorithm. When the algorithm terminates, Maverick returns the k context-subspace pairs with the highest exceptionality scores (Line 15). Below, we discuss component CE in Section 3.1, EE in Section 4, and PG in Section 5.

3.1 Context Evaluator

The context evaluator (CE, Line 8 in Alg. 1) is responsible for obtaining the matches to a given pattern as well as the corresponding contexts. Its working is depicted in Alg. 2. We expect a graph query system to take a pattern as the input and return all the matches to the pattern (Line 3). The Maverick framework is agnostic to the choice of the specific query processing system. According to Definition 2.4, for each variable in the pattern ($x \in X_P$), CE returns its range R^P_x as a context if the entity of interest v_0 is in the range (Line 5).

For example, consider graph G in Fig. 2, the entity of interest $v_0 =$ G1, and the pattern P_1 in Fig. 3a. $\mathcal{M}_{P_1} = \{M_1, M_2, M_3\}$, where M_1, M_2, and M_3 are in Figs. 3b–3d. P_1 has two variables, $?g$ and $?s$. Since G1 $\in R^{P_1}_{?g} = \{$G1, G2, G3$\}$ and G1 $\notin R^{P_1}_{?s}$, P_1 defines one and only one context of G1, which is $C^{P_1,?g}_{\text{G1}} = R^{P_1}_{?g}$. Therefore, $C^P_{\text{G1}} = \{C^{P_1,?g}_{\text{G1}}\}$.

4 EXCEPTIONALITY EVALUATOR

The Exceptionality Evaluator (EE) operates in the inner loop of the Maverick framework (function EXCEPTIONALITY-EVALUATOR (v_0, C, k, χ) at Line 10 of Alg. 1). For each context C of the entity of interest v, it finds the k subspaces A with the highest $\chi(v, A, C)$ scores. Note that it is sufficient to find these k subspaces, since the eventual output of Maverick is the top-k context-subspace pairs across all contexts of v. A naive solution of EE can exhaustively enumerate all possible subspaces of A_v and calculate the exceptionality score of v in each subspace. The apparent $O(2^{|A_v|})$ complexity of this approach renders it prohibitively expensive since many entities may have a lot of attributes. For instance, Denzel Washington has more than 40 attributes in the August 9, 2015 Freebase graph. It is thus crucial for Maverick to have an efficient subspace enumeration method in order to discover more exceptional context-subspace pairs. Section 4.1 discusses how Maverick uses a set enumeration tree to avoid exhaustively enumerating subspaces. Specifically, Maverick exploits the upper bound properties of exceptionality scoring functions to guide the traversal of the set enumeration tree. Section 4.2 introduces three representative exceptionality scoring functions along with their upper bound functions.

Algorithm 3: Exceptionality evaluator.

EXCEPTIONALITY-EVALUATOR $(v,\ C,\ k,\ \chi)$
    // CS: current subspace; UA: attributes to visit; Tk: top-k subspaces.
    CS $\leftarrow \emptyset$; UA $\leftarrow A_v$; Tk $\leftarrow \emptyset$;
    return EXPLORE-SUBSPACE(v, C, k, χ, CS, UA, Tk) ;
EXPLORE-SUBSPACE (v, C, k, χ, CS, UA, Tk)
    while UA $\neq \emptyset$ **do**
        // Calculate upper bounds.
        $a_{\max} \leftarrow \arg\max_{a \in \text{UA}}$ upper(v, CS $\cup \{a\}$, C);
        $A_{\max} \leftarrow$ CS $\cup \{a_{\max}\}$; upper$_{\max} \leftarrow$ upper(v, $A_{\max}$, C);
        UA $\leftarrow$ UA $\setminus \{a_{\max}\}$;
        if $|$Tk$| < k$ **then**
            // -1 indicates the top-k list Tk is not full.
            score$_{\min} \leftarrow -1$; $A_{\min} \leftarrow \emptyset$;
        else ($A_{\min}$, score$_{\min}$) $\leftarrow \arg\min_{(A,\text{score}) \in \text{Tk}}$ score ;
        if upper$_{\max}$ $>$ score$_{\min}$ **then**
            score $\leftarrow \chi(v, A_{\max}, C)$;
            if score $>$ score$_{\min}$ **then**
                if score$_{\min} \geq 0$ **then**
                   Tk $\leftarrow$ Tk $\setminus \{(A_{\min}$, score$_{\min})\}$;
                Tk $\leftarrow$ Tk $\cup \{(A_{\max}$, score$)\}$;
            // Explore children subspaces.
            Tk $\leftarrow$ EXPLORE-SUBSPACE(v, C, k, χ, $A_{\max}$, UA, Tk);
    return Tk;

4.1 Finding Top-k Subspaces

EE applies a *set enumeration tree* [22] to avoid exhaustively enumerating subspaces. Each node in the tree is a subspace—a subset of v's attributes A_v. The children of a node correspond to various supersets of the node's associated attributes. The gist is to explore the set enumeration tree using heuristic search methods such as best-first search and to prune branches that are guaranteed to not contain highly-scored subspaces.

What is particularly challenging is that an exceptionality scoring function χ usually does not have the *downward closure property* with respect to subspace inclusion, i.e., $\chi(v, A, C)$ can be greater than, less than, or equal to $\chi(v, A', C)$ for any $A' \supseteq A$. As a matter of fact, none of the three representative functions that will be introduced in Section 4.2 satisfies the property (proof omitted due to space limitations). The lack of downward closure property makes it infeasible to prune the set enumeration tree based on exact exceptionality scores.

EE uses upper bounds on the exceptionality scoring function χ to allow for pruning of the set enumeration tree. Alg. 3 presents its pseudo code. The set enumeration tree nodes (i.e., subspaces) are visited in the descending order of their upper bounds (Line 6). If the upper bound score of a node is not greater than the score of the current k-th ranked subspace, the node and all its children are pruned (Line 12). Otherwise, the exact exceptionality score of the node is calculated (Line 13). The subspace is used to purge the current k-th subspace if its exact score is still greater (Lines 14–17). Regardless of whether the node makes into the top-k list, its children are enumerated recursively (Line 18).

The general upper bound function *upper* in Alg. 3 is defined as follows. By the definition, it is sound to prune a node and all its children if the condition in Line 12 is not satisfied.

Definition 4.1 (Upper bound of an exceptionality scoring function upper). Given an exceptionality scoring function χ, a upper bound of χ is a function that, for any entity v, context C, and subspace $A \subseteq A_v$, bounds the exceptionality score of v with respect to C and any superset of A, i.e.,

$$upper(v, A, C) \geq \max_{A \subseteq A' \subseteq A_v} \chi(v, A', C). \qquad \triangle$$

The general upper bound function *upper* must be instantiated for specific exceptionality scoring functions χ. The Maverick framework expects an application developer to supply *upper* while specifying χ. Various outlying aspect mining methods [2, 3, 10] also devise upper bound functions for pruning set enumeration tree. They operate on the single-table data model and are thus inapplicable for graphs, as explained in Section 1. EE must use different scoring functions and upper bound functions designed for knowledge graphs. The ensuing discussion in this section entails that.

4.2 Exceptionality Scoring Functions

The Maverick framework is indifferent to the choice of the exceptionality scoring function. It can accommodate many different interestingness/outlyingness functions (see surveys such as [12, 18]). This section considers three representative functions. To ensure consistency, the discussion uses our own notations and terminologies in presenting the adaptation of existing functions.

Outlyingness χ_o This measure, adopted from [3], is based on the distribution of attribute values. An entity receives a high score if it has rare attribute values while a lot of other entities share common attribute values. Consider a context C, a subspace A, and any entity u in the context. We denote by p_S^A, or simply p_S when A is clear, the probability (or "frequency" as in [3]) of u taking attribute values S in subspace A, i.e.,

$$p_S^A = p(u.A = S \mid u \in C) = \mid \{u \mid u \in C,\ u.A = S\} \mid / \mid C \mid . \quad (1)$$

Let $\mathcal{S}_A$ be all possible attribute values on the subspace and in the context, i.e., $\mathcal{S}_A = \{u.A \mid u \in C\}$. The outlyingness score of an entity v is given by:

$$\chi_o(v, A, C) = \sum\nolimits_{S \in \mathcal{S}_A} p_S \times (p_S - p_{v.A}) \times \mathbb{1}(p_S > p_{v.A}) \quad (2)$$

where $\mathbb{1}(\cdot)$ is the indicator function that returns 1 for a true condition and 0 otherwise. Essentially, the outlyingness score is the area *above* the accumulated frequency histogram of the context C with respect to the subspace A, starting from the frequency of $v.A$. The score is designed to quantify the "degree of unbalance" between the frequencies of entities in the context [3].

Table 1: The probability distributions of attribute values in all subspaces for entity of interest G1 with regard to context C = {G1, G2, G3}.

A	$S \in \mathcal{S}_A$ and p_S^A	G1.A
{(*awarded-to*, →)}	⟨{CRO}⟩:1/3, ⟨{BRA}⟩:2/3	⟨{CRO}⟩
{(*scored-by*, →)}	⟨{S1}⟩:1/3, ⟨{S2}⟩:1/3, ⟨{S3}⟩:1/3	⟨{S1}⟩
{(*awarded-to*, →), (*scored-by*, →)}	⟨{CRO}, {S1}⟩:1/3, ⟨{BRA}, {S2}⟩:1/3, ⟨{BRA}, {S3}⟩:1/3	⟨{CRO}, {S1}⟩

For example, consider entity of interest v_0 = G1 in Fig. 2 and context C defined by pattern P_1 and variable ?g in Fig. 3a, i.e., $C = C_{\text{G1}}^{P_1,?g}$ = {G1, G2, G3}. Table 1 shows the probability distributions of attribute values in all subspaces. According to the table, χ_o(G1, {(*awarded-to*, →)}, C) = $p_{\langle\{\text{CRO}\}\rangle} \times (p_{\langle\{\text{CRO}\}\rangle} - p_{\langle\{\text{CRO}\}\rangle}) \times 0 + p_{\langle\{\text{BRA}\}\rangle} \times (p_{\langle\{\text{BRA}\}\rangle} - p_{\langle\{\text{CRO}\}\rangle}) \times 1 = \frac{2}{3}(\frac{2}{3} - \frac{1}{3}) = \frac{2}{9}$. Another example is, for A = {(*awarded-to*, →), (*scored-by*, →)}, χ_o(G1, A, C) = 0 since there exists no $S \in \mathcal{S}_A$ such that $p_S^A > p_{\langle\{\text{CRO}\},\{\text{S1}\}\rangle}^A$.

One-of-the-Few χ_f The one-of-the-few concept is adapted from [31]. The crux of the idea is that a factual claim about an entity is interesting when equally or more significant claims can be made about only few other entities. For example, in Fig. 2, it is interesting to claim "G1 is the only own goal among the goals scored by BRA players", since such a unique claim cannot be made about any other goal scored by a BRA player. On the contrary, "G1 is the only goal scored by S1" is not impressive, because the same kind of claim "Gx is the only goal scored by Sy" can be made for all 5 goals in Fig. 2.

The definition of one-of-the-few in [31] is based on multi-criteria dominance relationship which is irrelevant to this work. Our adaptation of [31] quantifies the rareness of attribute values by probability $p_S^A = p(u.A = S \mid u \in C)$ (the same as for χ_o) and the exceptionality of an entity's rareness by rank of the probability. Specifically, the exceptionality of an entity of interest v is given by:

$$\chi_f(v, A, C) = \mid \{u \mid u \in C,\ p_{u.A} > p_{v.A}\} \mid / \mid C \mid . \quad (3)$$

For instance, consider the same example used in explaining χ_o: v_0 = G1, $C = C_{\text{G1}}^{P_1,?g}$ = {G1, G2, G3}, and A = {(*awarded-to*, →)}. According to Table 1, $p_{\text{G2}.A} = p_{\text{G3}.A} = p_{\langle\{\text{BRA}\}\rangle} = \frac{2}{3} > p_{\text{G1}.A} = p_{\langle\{\text{CRO}\}\rangle} = \frac{1}{3}$. Hence, $\chi_f(\text{G1}, A, C) = \frac{\mid \{\text{G2}, \text{G3}\} \mid}{\mid C \mid} = \frac{2}{3}$. For A = {(*awarded-to*, →), (*scored-by*, →)}, χ_f(G1, A, C) = 0, since there exists no $u \in C$ such that $p_{u.A} > p_{\text{G1}.A}$.

Outlyingness Rank χ_r The outlyingness rank $\chi_r(v, A, C)$ is a generalization of χ_f. It is based on an outlyingness degree function δ, as follows:

$$\chi_r(v, A, C) = \mid \{u \mid u \in C,\ \delta(u, A, C) > \delta(v, A, C)\} \mid / \mid C \mid . \quad (4)$$

[10] uses probability p_S^A for δ in which χ_r becomes χ_f. An alternative instantiation of δ is based on the distance between v and its k-th nearest neighbor [17, 20], capturing to what degree v stands out from other entities. For instance, let δ be the negation of the distance between v and its nearest neighbor (i.e., $k = 1$):

$$\delta(v, A, C) = -\min_{u \in C} d(u, v, A) \quad (5)$$

where the distance measure is the L^2 norm of the attribute values' *Jaccard dissimilarity*:

$$d(u, v, A) = \sqrt{\sum_{a \in A} \left(1 - \frac{\mid u.a \cap v.a \mid}{\mid u.a \cup v.a \mid}\right)^2}. \quad (6)$$

For example, Table 2 shows the distances between entities in C={G1, G2, G3} by Eq. (6). In subspace A={(*awarded-to*, →)}, the nearest neighbor of G1 is either G2 or G3. Therefore the distance between G1 and its nearest neighbor is 1 and δ(G1, A, C) = −1. Similarly,

Table 2: The distances between entities in {G1, G2, G3} based on the distance function Eq. (6).

A	d(G1, G2, A)	d(G1, G3, A)	d(G2, G3, A)
{(*awarded-to*, →)}	1	1	0
{(*scored-by*, →)}	1	1	1
{(*awarded-to*, →), (*scored-by*, →)}	$\sqrt{2}$	$\sqrt{2}$	1

G2 and G3 are the nearest neighbors of each other, and δ(G2, A, C) = δ(G3, A, C) = 0. Based on these values, χ_r(G1, {(*awarded-to*, →)}, C) = $\frac{|\{\text{G2,G3}\}|}{|C|} = \frac{2}{3}$.

4.3 Upper Bound Functions

In this section we devise upper bound functions for the three representative exceptionality functions introduced in Section 4.2. We prove that these designs satisfy Definition 4.1 and thus ensure the soundness of Alg. 3, with regard to any given entity v, context C, and subspaces $A \subseteq A' \subseteq A_v$. Recall that we denote by p_S^A, or simply p_S, the probability of an entity taking attribute values S in subspace A (Eq. (1)).

THEOREM 4.2 (UPPER BOUND OF χ_o). *$upper_o(v, A, C) \geq \chi_o(v, A', C)$, given the following definition where $\mathcal{S}_A = \{u.A \mid u \in C\}$:*

$$upper_o(v, A, C) = \sum\nolimits_{S \in \mathcal{S}_A} (p_S)^2 - \frac{(2\,p_{v.A} + 1) \times |C| - 2}{|C|^2}. \quad (7)$$

PROOF. Let $\{p_{v.A}, p_{S_1}, \cdots, p_{S_N}\}$ be the probability distribution of attribute values in subspace A. According to [3], for any $A' \supseteq A$, $\chi_o(v, A', C)$ is maximized when the additional attributes in $A' \setminus A$ preserve the current attribute value distribution, except that the additional attributes make v different from all other entities, i.e., the optimal distribution of attribute values in subspace A' is $\{p_{v.A'}, p_{v.A} - p_{v.A'}, p_{S_1}, \cdots, p_{S_N}\}$, where $p_{v.A'} = \frac{1}{|C|}$. (Note that $p_S \geq \frac{1}{|C|}$ for any S.) In other words, the entities having value $v.A$ on subspace A are partitioned into v itself (having value $v.A'$ on subspace A') and the rest (having identical value on A'). Based on Eq. (2), after a few polynomial manipulations, which we omit here, we have $\chi_o(v, A', C) \leq \sum_{S \in \mathcal{S}_A} {p_S}^2 - \frac{(2\,p_{v.A}+1) \times |C| - 2}{|C|^2}$. ■

THEOREM 4.3 (UPPER BOUND OF χ_f). *$upper_f(v, A, C) \geq \chi_f(v, A', C)$, given the following definition in which $\overline{C_v} = C \setminus \{v\}$:*

$$upper_f(v, A, C) = |\{u \mid u \in \overline{C_v},\ p_{u.A}^A > 1/|C|\}| \Big/ |C| \quad (8)$$

The theorem holds because $\frac{1}{|C|} \leq p_{u.A'} \leq p_{u.A}$ for any $A' \supseteq A$. We omit the detailed proof here.

THEOREM 4.4 (UPPER BOUND OF χ_r). *$upper_r(v, A, C) \geq \chi_r(v, A', C)$, given the following definition of upper bound:*

$$upper_r(v, A, C) = \frac{|\{u \mid u \in \overline{C_v},\ \delta_{\max}(u, A, C) > \delta_{\min}(v, A, C)\}|}{|C|} \quad (9)$$

where δ is the outlyingness degree function, and $\delta_{\max}$ and $\delta_{\min}$ are the upper and lower bounds of δ, respectively.

Theorem 4.4 holds because Eq. (9) uses the best δ value for v and the worst value for other entities. (Note that the lower δ is, the more outstanding the entity is, according to Eq. (5).) We omit the detailed proof. We propose the following instantiations of $\delta_{\max}$ and $\delta_{\min}$ for the δ defined by Eq. (5), and we prove they are valid upper and lower bounds (see Appendix A.3).

$$\delta_{\max}(v, A, C) = \delta(v, A, C)$$
$$\delta_{\min}(v, A, C) = -\sqrt{\delta(v, A, C)^2 + (|A_v| - |A|)}$$

Our final note is that an upper bound function may have limited pruning power when it gives loose bounds on exceptionality scores, resulting in exponential complexity in subspace enumeration. Our empirical results in Section 6.2, though, verified that the several upper bound functions proposed above (Eqs. (7)–(9)) substantially reduced the overhead of subspace enumeration.

5 PATTERN GENERATOR

The Pattern Generator (PG) is used in Line 12 of Alg. 1 in the Maverick framework. Its pseudo code is in Alg. 4. At each iteration of the beam search on patterns, it finds the children of each visited pattern P (Line 3, see Alg. 5) in the current beam. A child pattern, if not pruned (see Section 5.3), is given a score that measures how promising it is according to a few heuristics (Line 5, see Section 5.4). Among all the children of the patterns in the current beam, the w children with the highest scores are returned to form the new beam (Line 14 in Alg. 1), where w is the predefined beam width. The new beam becomes the input to the next iteration. This section first describes the search space of patterns (Section 5.1) and then discusses how to efficiently explore the space by applying pruning rules (Section 5.3) and selection heuristics (Section 5.4).

Algorithm 4: Pattern generator.

```
1 PATTERN-GENERATOR (v0, P, M_P, w, G)
2     Y ← ∅ ;                                   // Promising children of P.
      // Find P's children, see Alg. 5.
3     children ← FIND-CHILDREN(v0, P, M_P, G);
4     foreach child ∈ children do
5         score ← h(v0, child);                 // See Section 5.4 for h.
6         Y ← Y ∪ {(child, score)};
7     return top-w of Y based on score;
```

5.1 Search Space of Patterns

The search space of patterns is a Hasse diagram of valid patterns, where a pattern is *valid* if it contains at least one variable node and it has a match (Definition 2.2) in the knowledge graph G. We exclude invalid patterns since they cannot lead to relevant facts. For example, pattern {(?g, *scored-by*, ?s1), (?g, *scored-by*, ?s2)} does not have a match and is thus invalid because no goal is scored by more than one player. Formally, the search space of patterns is a Hasse diagram $\mathbb{P}(V_{\mathbb{P}}, E_{\mathbb{P}})$, where $V_{\mathbb{P}}$ is the set of valid patterns and $E_{\mathbb{P}} \subseteq V_{\mathbb{P}} \times V_{\mathbb{P}}$ is the set of edges. There exists an edge from *parent* pattern P_i to *child* pattern P_j if P_i is an immediate subgraph of P_j, i.e., P_i has exactly one edge less than P_j. A pattern can have multiple children and multiple parents. Fig. 5 shows an excerpt of the search space of patterns over the data graph in Fig. 2. In the figure, P_6 and P_7 are the children of P_2, and both P_2 and P_3 are the parents of P_7.

One may realize already that $\mathbb{P}$ can be extremely large. We prove in Theorem 5.1 that the *order* of $\mathbb{P}$ (i.e., the cardinality of $V_{\mathbb{P}}$) is exponential to the orders of G's weakly connected components (WCCs).[4] Given that knowledge graphs are all well connected, it is impossible to exhaustively enumerate the patterns. For example, according to Theorem 5.1, the data graph in Fig. 2 has at least

[4] A weakly connected component is a maximal subgraph of a directed graph, in which every pair of vertices are connected, ignoring edge direction.

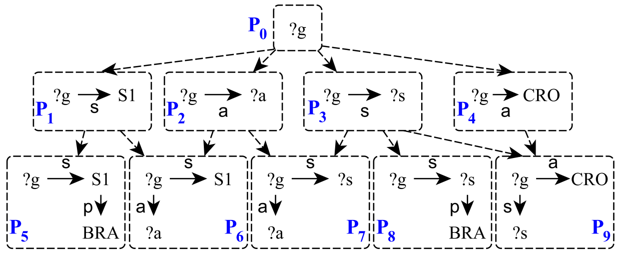


Figure 5: An excerpt of the search space of patterns over Fig. 2. Edge labels: *a*: *awarded-to*, *p*: *play-for*, *s*: *scored-by*.

$2^{13+1} - 2 - 13 + 13 = 16,382$ patterns. (The graph itself is the only WCC, with 13 nodes.) Note that Theorem 5.1 only provides a loose bound. In practice, the number is even much larger, exacerbating the challenge. Section 6.2 shows that the tiny graph has more than 69,000 patterns with merely no more than 5 edges.

THEOREM 5.1. *Let $\mathcal{W}$ be the set of WCCs in a knowledge graph G, a lower bound on $\mathbb{P}$'s order is:*

$$| V_{\mathbb{P}} | \geq \sum_{W \in \mathcal{W}} (2^{| V_W |+1} - 2) - | V_G | + \max_{W \in \mathcal{W}} | V_W |.$$

PROOF. Given a WCC $W \in \mathcal{W}$, it has at least one subgraph of order i, for every $i \in [\ 1, |V_W|\ ]$. For each subgraph of size i, there are 2^i corresponding patterns that can be constructed by replacing some nodes with variables. Hence, for each W, there are at least $\sum_{i=1}^{|V_W|} 2^i = 2^{|V_W|+1} - 2$ patterns. Since every such pattern is isomorphic to a subgraph of W, it is guaranteed to be valid. Note that two patterns of the same order constructed from two subgraphs in two different WCCs can be equivalent if all their nodes are variables. Therefore, each $W \in \mathcal{W}$ has at most $| V_W |$ patterns that are equivalent to others. There are at least $\max_W |V_W|$ unique patterns in which all nodes are variables. Thus, after excluding double-counted patterns,

$$\begin{aligned} | V_{\mathbb{P}} | &\geq \sum_{W} (2^{|V_W|+1} - 2) - \sum_{W} |V_W| + \max_{W} |V_W| \\ &= \sum_{W} (2^{|V_W|+1} - 2) - |V_G| + \max_{W} |V_W|. \end{aligned}$$ ∎

5.2 Match-based Construction of Patterns

Given the current beam of patterns, Maverick finds top context-subspace pairs using its context evaluator (Section 3.1) and exceptionality evaluator (Section 4). Among the child patterns of the evaluated patterns, the promising ones are chosen to form the new beam for the next iteration. While Section 5.4 discusses how to select the promising patterns, this section proposes an efficient way of generating the child patterns. Note that the aforementioned Hasse diagram of patterns is not pre-materialized. Rather, the patterns need to be constructed before we can evaluate them.

To construct the child patterns of an evaluated pattern P, a simple approach is to enumerate all possible ways of expanding P by adding one more edge. A major drawback of this approach is it may construct many invalid patterns that do not have any match. Some invalid patterns can be easily recognized by referring to the schema graph of the data. However, chances are most of the schema-abiding patterns are still invalid because they do not have matching instances in the data graph, given the sheer diversity of a knowledge graph. The system will evaluate such patterns in vain to get empty results in order to realize they are invalid.

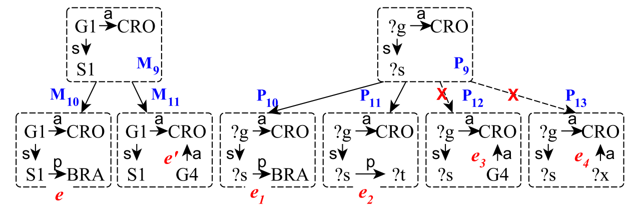


Figure 6: Illustration of how the child patterns of a pattern are constructed. P_{10} and P_{11} are obtained based on M_{10} and edge e. P_{12} and P_{13} can be obtained based on M_{11} and edge e', but they are pruned based on rules in Section 5.3.

To avoid evaluating invalid patterns, we propose a *match-based pattern construction method*. Instead of constructing the child patterns by directly expanding P, this method expands the matches of P and constructs the child patterns from the expanded matches. It guarantees to construct only valid patterns and evade the evaluation of invalid patterns. The method is based on the following theorem.

THEOREM 5.2. *Suppose P' is a child of $P \in \mathbb{P}$, i.e., $(P, P') \in E_{\mathbb{P}}$ and thus P' is a valid pattern with matches. Given any match M' to P', there exists a match M to P that is a subgraph of M', i.e., $\forall M' \in \mathcal{M}_{P'}, \exists M \in \mathcal{M}_P$ s.t. $V_M \subseteq V_{M'}$ and $E_M \subseteq E_{M'}$.*

PROOF. Since P' is a child of P, P' has one edge more than P. Suppose $(u, l, w) = E_{P'} \setminus E_P$, and f' is the bijection $f' : V_{P'} \to V_{M'}$. We prove the theorem by constructing M. More specifically, let $E_M = E_{M'} \setminus \{(f'(u), l, f'(w))\}$, and $V_M = \cup_{(v_i, l', v_j) \in E_M} \{v_i, v_j\}$. We can construct a bijection $f : V_P \to V_M$ such that $f(u) = f'(u)$ for any $u \in V_P$. Since f' satisfies the edge isomorphism, f also satisfies it, i.e., $\forall (v_i, l', v_j) \in E_P$, $(f(v_i), l', f(v_j)) \in E_M$, and vice versa. By Definition 2.2, M is a match to P. ∎

Based on Theorem 5.2, the method that constructs the child patterns of P is illustrated in Alg. 5. For a match M of P, it finds each of its weakly connected supergraphs by adding an edge that exists in the data graph G and is adjacent to a node in M (Line 6). Given each such resulting supergraph M', let $(u, l, w) = E_{M'} \setminus E_M$ and, without loss of generality, assume $u \in V_M$. If $w \in V_M$, then the only child of P obtained from M' is $P + (f^{-1}(u), l, f^{-1}(w))$ (Line 13).[5] If $w \notin V_M$, then two child patterns are obtained: $P + (f^{-1}(u), l, w)$ and $P + (f^{-1}(u), l, z)$, where z is a variable and $z \notin X_P$ (Line 17; Line 21 for the symmetric case). Fig. 6 shows an example of obtaining a pattern's children. For instance, P_{10} can be obtained by adding e_1, which is obtained by replacing S1 of edge *e* with variable ?s.

5.3 Pattern Pruning Strategies

The search space $\mathbb{P}$ of patterns as defined in Section 5.1 and constructed using the match-based pattern construction method in Section 5.2 has an enormous size. To ensure efficiency, the Pattern Generator (PG) employs two pruning rules to exclude irrelevant patterns from $\mathbb{P}$ and to avoid repeated constructions of patterns from certain type of parent patterns.

Rule 1 (*RelevantOnly*). Exclude a pattern if it does not define any context for the entity of interest v_0.

The rational behind Rule 1 is, for discovering exceptional facts about v_0, a pattern is relevant only if it defines a context for v_0. By this rule, the match-based pattern construction method only expands

[5] For brevity, we denote by $P + e$ the supergraph of P by adding edge e.

Algorithm 5: Find all the children of a given pattern.

FIND-CHILDREN $(v_0, P, \mathcal{M}_P, G)$
  $D \leftarrow \varnothing$; // The set of P's children.
  $\mathbb{M} \leftarrow \{M \in \mathcal{M}_P \mid f{:}V_P \rightarrow V_M$ and $\exists x \in X_P$ s.t. $f(x) = v_0\}$; // Rule 1
  foreach $M \in \mathbb{M}$ **do**
    Let f be the bijection $f : V_P \rightarrow V_M$;
    $\overline{E_M} = \{(u, l, w) \in E_G \setminus E_M \mid u \in V_M$ or $w \in V_M\}$;
    foreach $(u, l, w) \in \overline{E_M}$ **do**
      $z \leftarrow$ a new variable and $z \notin X_P$;
      if $\nexists\, x \in X_P$ *s.t.* $f(x) = u$ *or* $f(x) = w$ **then**
        continue; // Rule 2
      else if $u \in V_M$ *and* $w \in V_M$ **then**
        $x \leftarrow f^{-1}(u)$, $y \leftarrow f^{-1}(w)$;
        $P_1 \leftarrow P + (x, l, y)$;
        $D \leftarrow D \cup \{P_1\}$;
      else if $w \notin V_M$ **then** // $\exists\, x \in X_P$ s.t. $f(x) = u$
        $x \leftarrow f^{-1}(u)$;
        $P_1 \leftarrow P + (x, l, w)$; $P_2 \leftarrow P + (x, l, z)$;
        $D \leftarrow D \cup \{P_1, P_2\}$;
      else // $\exists\, y \in X_P$ s.t. $f(y) = w$
        $y \leftarrow f^{-1}(w)$;
        $P_1 \leftarrow P + (u, l, y)$; $P_2 \leftarrow P + (z, l, y)$;
        $D \leftarrow D \cup \{P_1, P_2\}$;
  return D;

(1) (x, l, y)	(2) (x, l, z)	(3) (x, l, w)	(4) (u, l, y)
(5) (u, l, z)	(6) (u, l, v)	(7) (u, l, w)	

Figure 7: Consider a pattern P and its child pattern P'. The 7 types of the extra edge $e = E_{P'} \setminus E_P$. x, y, z are variables, $x, y \in X_P$, $z \notin X_P$. u, v, w are non-variables, $u, v \in Y_P$, and $w \in V_G \setminus Y_P$.

a match in which v_0 is an image of a variable in P. It is guaranteed that the patterns obtained define v_0's contexts.

Rule 2 (*VarOnly*)**.** Expand a pattern only if the new edge has at least one variable.

Let P' be a child pattern of P. The extra edge in P', i.e., $e{=}E_{P'}\setminus E_P$, can belong to one of the 7 types in Fig. 7. The construction of P' from P is avoided by Rule 2 if e belongs to types 6-7. This rule is based on Theorem 5.3. Simply put, enforcing Rule 2 will not miss any contexts of v_0.

THEOREM 5.3. *Let P' be a child of $P \in \mathbb{P}$, $e = E_{P'} \setminus E_P$, $C^P_{v_0}$ be all the contexts of v_0 defined by P: $C^P_{v_0} = \{R^P_{x'} \mid x' \in X_P, v_0 \in R^P_{x'}\}$, then $C^{P'}_{v_0} = C^P_{v_0}$, if e belongs to types 6-7.*

PROOF. Since both ends of e are entities, we have $X_P = X_{P'}$. By Theorem 5.2, $\forall M' \in \mathcal{M}_{P'}$, there exists $M \in \mathcal{M}_P$ which is a subgraph of M'. Let $e' = E_{M'} \setminus E_M$, then $e' = e$ by Definition 2.2. Therefore, $\forall x \in X_{P'}$, $R^{P'}_x \subseteq R^P_x$. Similarly, $\forall M \in \mathcal{M}_P$, the graph $M + e$ is a match to P'. As a result, $\forall x \in X_P$, $R^P_x \subseteq R^{P'}_x$. In sum, $\forall x \in X_P$, $R^P_x = R^{P'}_x$, and $C^P_{v_0} = \{R^P_{x'} \mid x' \in X_P, v_0 \in R^P_{x'}\} = \{R^{P'}_{x'} \mid x' \in X_{P'}, v_0 \in R^{P'}_{x'}\} = C^{P'}_{v_0}$. ∎

5.4 Pattern Selection Heuristics (h)

Even with the rules proposed in Section 5.3, there are still too many patterns. In this section, we propose two scoring heuristics for selecting promising patterns to visit, to substantiate the function h in Line 5 of Alg. 4. A heuristic gives each pattern a score, based on which the w patterns with the highest scores form the beam for the next iteration of beam search.

Heuristic 1 (Optimistic)**.** Given a pattern P, the entity of interest v_0, let $C^P_{v_0}$ be the set of contexts defined by P, i.e., $C^P_{v_0} = \{C^{P,x}_{v_0} \mid x \in X_P, v_0 \in R^P_x\}$, then

$$h_{opt}(v_0, P) = \max_{C \in C^P_{v_0}} upper(v_0, \varnothing, C)$$

where $upper(v_0, \varnothing, C)$ is a upper bound of χ with regard to C for any subspace (see Defintion 4.1).

h_{opt} simply uses the exceptionality score upper bound of P. It optimistically assumes the ideal case for each pattern, where the entity of interest is most exceptional among the entities in a context defined by the pattern. In Section 4, we discussed the upper bound functions for various exceptionality functions. Note that we have $p_{v_0.\varnothing} = 1$ (Eq. (1)) since $v.\varnothing =$ `null` (Definition 2.6) and we consider all null values equal, $upper_o(v_0, \varnothing, C) = 1 - \frac{3\times|C|-2}{|C|^2}$, and $upper_f(v_0, \varnothing, C) = 1 - \frac{1}{|C|}$, according to Eq. (7) and Eq. (8), respectively. Similarly, according to Eq. (5) and Eq. (6), $\forall v \in C$, $\delta(v, \varnothing, C) = 0$, and $\delta_{\max}(v, \varnothing, C) = 0$, $\delta_{\min}(v, \varnothing, C) = -\sqrt{|A_v|}$. Thus according to Eq. (9), $upper_r(v_0, \varnothing, C) = 1 - \frac{1}{|C|}$. In sum, all the three upper bounds increase when the context size increases. In other words, h_{opt} selects the patterns that define large contexts. However, a large context may contain many entities of different characteristics, which may actually make the entity of interest less exceptional. Note that, since h_{opt} depends on context size $|C|$, all the child patterns of P need to be evaluated in order to get $|C|$. It is also required for heuristic h_{conv} below for the same reason.

Heuristic 2 (Convergent)**.** Consider a pattern P and the entity of interest v_0. Given P', a parent of P in the pattern search tree, we define $r_x = |C^{P,x}_{v_0}| \big/ |C^{P',x}_{v_0}|$. The score of P is

$$h_{conv}(v_0, P) = \max_{(P',P)\in E_{\mathbb{P}} \text{ and } P' \in B, C^{P',x}_{v_0} \in C^{P'}_{v_0}} \Big[r_x \times \max_{A \subseteq A_{v_0}} \chi(v_0, A, C^{P',x}_{v_0}) + (1 - r_x) \times upper(v_0, \varnothing, C^{P,x}_{v_0}) \Big]$$

The h_{conv} score of P is a weighted sum of the upper bound of P (for any subspace) and the best score of the parent pattern P'. Note that Maverick performs a beam search and the patterns visited form a pattern search tree. P could be constructed from different parent patterns in the current beam B. The above equation thus uses the best score across all such parents. For this reason, the edge adjacent to P in the pattern search tree comes from the parent P' that gives it the best score. If P' posses some highly-scored context-subspace pairs, h_{conv} gives favorable score to P if P and P' define similar contexts; otherwise, h_{conv} favors a P that defines smaller contexts. Compared with h_{opt}, h_{conv} is potentially both more efficient and more effective. It can be more efficient since it may favor child patterns that define smaller contexts. Such child patterns usually can be evaluated more efficiently since they have less matches. It can be more effective since it discards child patterns that define contexts where the entity of interest may not be exceptional, based on the highest score of the context-subspace pairs for the parent pattern. When h_{conv} is used for choosing patterns to form the beams, the sizes of the contexts defined by the patterns in a path of the tree may gradually become smaller and eventually converge. We thus call h_{conv} Convergent.

6 EXPERIMENTS

6.1 Experiment Setup

The framework and algorithms of Maverick are implemented in Python. The experiments were conducted on a 16-core, 32GB-RAM node in Stampede—a cluster of the Extreme Science and Engineering Discovery Environment (XSEDE: *https://www.xsede.org*).

Datasets The experiments used the following two real-world graphs:

- WCGoals. It was constructed by crawling data from the FIFA World Cup website (*http://www.fifa.com/worldcup/index.html*). It consists of 49,078 nodes, 158,114 edges, 13 different edge labels, and 11 entity types: WorldCup, RoundCategory, Round, Stadium, Team, Game, Group, Player, BibNum, Participant, and Goal.
- OscarWinners. This is a subgraph of Freebase. It has 42,148 nodes, 63,187 edges, 24 distinct edge labels, and 13 entity types including Person, FilmCrew, AwardWon, FilmCharacter, AwardCategory, Performance, Genre, Award, Film, Country, FilmCrewRole, Ceremony, and SpecialPerformanceType. Each film in the graph has won at least one Academy Award (Oscar).

The two graphs were stored using Neo4j (*https://neo4j.com*) graph database. The patterns are expressed in Neo4j's query language Cypher. The experiment results using the two graphs and different expectionality scoring functions are similar. Therefore, we only report our findings on WCGoals and exceptionality function χ_o, except that Section A.2 reports the discovered exceptional facts using both WCGoals and OscarWinners.

Methods Compared The experiments compared the performance of a breath-first search method and the beam search method (Section 3) coupled with different heuristics (Section 5.4). The compared methods are:

- Beam-Rdm: Beam search that randomly selects child patterns.
- Beam-Opt: Beam search using h_{opt} in selecting child patterns.
- Beam-Conv: Beam search using h_{conv} in selecting child patterns.
- Breath-First: The breath-first search method that enumerates all possible patterns.

The family of beam search methods and Breath-First differ in two ways. *Firstly*, beam search only visits a fixed number of patterns at each level of the pattern search tree, whereas Breath-First visits all. *Secondly*, beam search visits the patterns by the decreasing order of their scores, whereas Breath-First does not assume any order. The experiment results establish that, even though Breath-First may evaluate more patterns than the beam search methods in a fixed time frame, it is not as effective as Beam-Conv which discovers more highly-scored context-subspace pairs using less time.

6.2 Efficiency

We measured how fast Maverick discovers highly-scored context-subspace pairs and how fast it explores the search space of patterns. We executed Maverick for multiple 2-minute runs and recorded a) the scores of discovered context-subspace pairs; b) the time when each context-subspace pair was discovered; and c) the number of visited patterns in the outer loop of the framework.

Fig. 8 shows the heat map of the context-subspace pairs' exceptionality scores by their timestamps. It includes all the discovered context-subspace pairs during the 2-minute runs for 10 entities of interest in WCGoals. We run 10 times per entity for all the methods, since Beam-Rdm selects child patterns randomly. Both the output

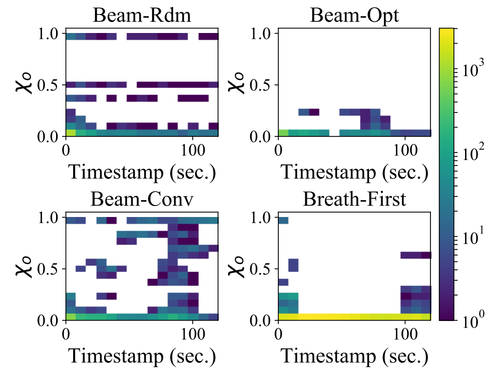
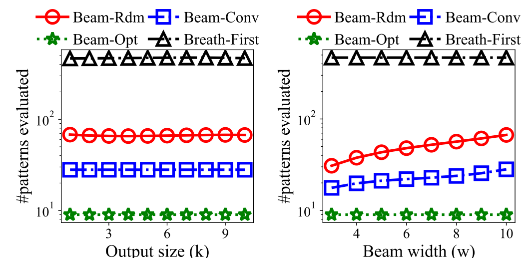


Figure 8: The heat map of exceptionality scores (χ_o) and timestamps of all the discovered context-subspace pairs during 2-minute runs for 10 entities of interest (v_0) in WCGoals ($k = 10$, $w = 10$).



a Varying k, fixing $w = 10$. **b Varying w, fixing $k = 10$.**

Figure 9: Effect of k and w on the number of evaluated patterns.

size k and the beam width w were set to 10. The 10 entities were randomly chosen from those that have highly-scored context-subspace pairs. Each bucket in the figure corresponds to a particular range of scores and a 8-second time frame in the 2-min run. The color of the bucket reflects how many context-subspace pairs (from all 100 runs for the 10 entities) discovered during the time frame fall into the corresponding score range. Intuitively, if the upper left portion of a heat map is more populated, the corresponding method performs better, since it means the method discovers highly-scored pairs faster. If the upper portion of a heat map is more populated, it means the method discovers more highly-scored pairs. The figure shows that Beam-Conv is both efficient and effective in discovering highly-scored context-subspace pairs. In contrast, Beam-Opt performed poorly. The results confirm the analysis in Section 5.4: preferring patterns that produce large contexts (h_{opt}) degrades not only the efficiency but also the effectiveness of Maverick. It is because such patterns are usually more expensive to evaluate and the produced contexts may include more varieties of entities, which makes the entity of interest less exceptional. With regard to Breath-First, since it enumerates candidate patterns exhaustively, it may discover some highly-scored pairs that reside in the low levels of the pattern search tree. For example, some highly-scored pairs for entity Goal(46683) were found using the 2-edge pattern in Fig. 3a. Given that the number of patterns with no more than 2 edges is small (more details in the discussion of results regarding pruning strategies), Beam-Rdm is likely to hit such small patterns that define contexts in which the entity of interest is exceptional.

Fig. 9 shows the impact of output size k and beam width w on how many patterns Maverick can manage to evaluate. The y-axis is the average number of evaluated patterns across the aforementioned

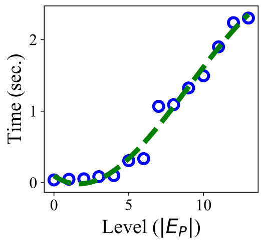


Figure 10: Time spent by Context Evaluator (Alg. 2) on evaluating patterns at different levels.

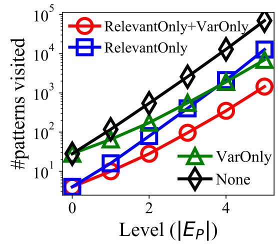
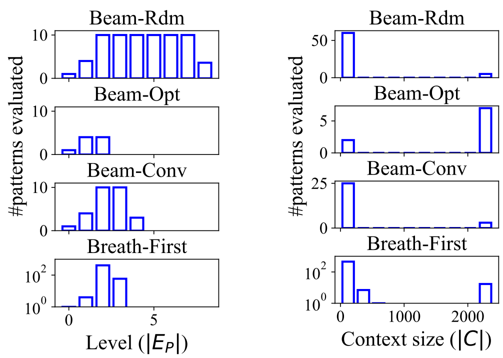


Figure 11: The pruning power of different pruning strategies.



a By level. **b By context size.**

Figure 12: Number of evaluated patterns by level and context size.

10 runs. Since we observed similar results on the 10 entities from WCGoals, the figure only depicts the results on Goal(46683). Fig. 9a shows that varying k from 1 to 10 (fixing w at 10) barely had any impact on the number of evaluated patterns. Since k controls the number of context-subspace pairs that Maverick returns, it mainly affects EE, which is responsible for finding top-k subspaces with regard to each context. Thus k needs to be very large in order to have a significant impact on the number of evaluated patterns, since EE is the least time-consuming component, as explained as follows. Table 3 provides the breakdown of execution time of different search methods into the three components in the workflow—Context Evaluator (CE), Exceptionality Evaluator (EE), and Pattern Generator (PG). The results are the average of the runs which are the same as in Fig. 8. (The summation in each column is slightly less than 100%, since we do not include operations such as framework initialization in the breakdown.) Another observation from Table 3 is that the execution time of PG dominates more substantially in Beam-Opt and Beam-Conv than in Beam-Rdm and Breath-First. The reason is PG in both Beam-Opt and Beam-Conv needs to compute h for each child pattern based on the pattern selection heuristics, which entails evaluating the child patterns to obtain the context sizes. In fact, on average, PG in Beam-Opt and Beam-Conv spent more than 99% and 96% of its time on applying the heuristics.

Table 3: Breakdown of execution time by components.

	Beam-Rdm	Beam-Opt	Beam-Conv	Breath-First
CE	25.52%	1.56%	1.90%	28.36%
EE	0.41%	0.65%	0.32%	2.79%
PG	61.49%	97.69%	95.92%	53.89%

Fig. 9b depicts the results when w varied from 3 to 10 and k was fixed at 10. It shows the number of evaluated patterns increased by w in the three beam search methods. When w increases, the methods evaluate more patterns from lower levels in the pattern search tree, which have less edges and can be evaluated more efficiently than those from higher levels. In a fixed time frame, the methods can then evaluate more patterns in total, as shown in the figure. Fig. 10 shows the average time that Context Evaluator (Alg. 2) spends on pattern evaluation increases when the level of pattern (i.e., the number of edges) increases. Since Breath-First does not need to calculate scores for patterns and does not have a limit on the number of patterns to visit at each level, it tends to evaluate more patterns but may only evaluate patterns at low levels. Fig. 12a compares the numbers of patterns evaluated at different levels by the four methods, when both k and w stayed at 10. Breath-First evaluated patterns up to level 3 and spent most of its time on level 3. On the contrary, the beam search methods evaluated at most 10 patterns at each level and covered more levels.

Fig. 9b also suggests that Beam-Rdm evaluated more patterns than Beam-Conv and Beam-Opt. It is because Beam-Rdm (like Breath-First) does not compute scores for child patterns, which is an expensive operation. Since Beam-Opt favors patterns that define larger contexts, it evaluated the fewest patterns since it spent more time to calculate the sizes of the contexts. On the other hand, Beam-Conv prefers patterns defining smaller contexts, which allowed it to evaluate more patterns. This is verified in Fig. 12b, which shows the numbers of evaluated patterns with different context sizes, when k and w were both 10.

Effect of pruning strategies We examined the effectiveness of the two pruning rules from Section 5.3 by comparing the following pruning strategies. In order to comprehensively compare these strategies, we used Breath-First as the search method since it exhaustively enumerates all possible candidate patterns at all levels.

- None: No child pattern pruning rule is applied;
- RelevantOnly (Rule 1);
- VarOnly (Rule 2);
- RelevantOnly+VarOnly: Apply both *RelevantOnly* and *VarOnly*.

Fig. 11 shows the number of patterns visited by Breath-First on the data graph in Fig. 2. The figure reveals that both rules can significantly reduce the number of candidate patterns. For instance, there are $69,582$ candidate patterns at level 5 when no pruning rule is applied (*None*). The number is reduced to $12,740$ and $6,963$ by following *RelevantOnly* and *VarOnly*, respectively. It is further reduced to $1,448$ with both rules applied (*RelevantOnly+VarOnly*). The figure also shows that the number of patterns still grows exponentially to the level of the pattern search tree even with both pruning rules applied, which suggests an enormous search space of patterns. Since *VarOnly* is stricter than *RelevantOnly*, as *VarOnly* only allows expanding on variable nodes, the growth rate of *VarOnly* can be smaller than *RelevantOnly*. Fig. 11 also confirms that.

Effect of upper bound of exceptionality functions Fig. 13 depicts the effect of using upper bound functions in pruning subspaces (Section 4.3). It shows the time and the number of subspaces visited for Game(903) which is one of the 10 entities used in Fig. 8, with/without applying upper bound functions under varying k. (Results for the other 9 entities are similar.) The measures are averages

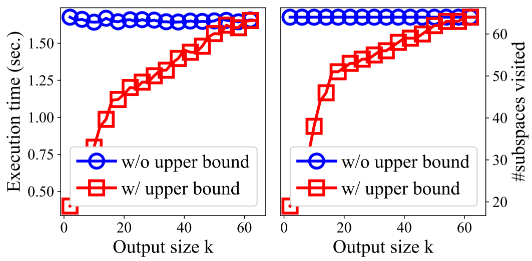


Figure 13: Effect of subspace pruning (upper bound functions).

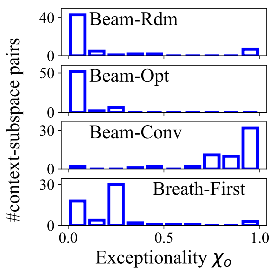


Figure 14: Score distributions of the top-10 exceptional context-subspace pairs for 10 entities, 10 2-minute runs per entity.

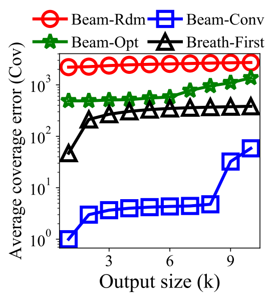


Figure 15: The average coverage error of each method on 10 entities (beam width $w = 10$).

of 10 runs. The figure verifies that the upper bound functions significantly improve the performance of exceptionality score calculations. Under relatively small k (e.g. 10), the execution time was reduced by more than half when the upper bound was applied. As k increased, the upper bound function's pruning power gradually diminished. Eventually, it was no longer able to prune any subspaces after $k = 60$.

6.3 Effectiveness

We also conducted experiments to verify if Maverick can effectively discover highly-scored context-subspace pairs. Fig. 14 shows the score distributions of the top-10 context-subspace pairs for the same 10 entities used in Section 6.2. There were 10 runs per entity and each run was 2-minute. Both parameters k and w were set to 10. The results in Fig. 14 are averaged over all entities and all runs. The results show that the output of Beam-Rdm mainly consists of pairs scored low. It is expected because the chance of hitting a promising pattern by a random method is very low due to the large search space of patterns. It is not surprising either to observe Beam-Opt performed badly as explained in Section 6.2. In contrast, Beam-Conv significantly outperformed other beam search methods, as it found much more highly-scored context-subspace pairs. It also found substantially more highly-scored pairs than Breath-First in score range $[0.8 - 1.0]$. This observation confirms that a wide pattern search tree hinders Breath-First's performance.

We also use a variation of *coverage error* [26] to measure the effectiveness of the four methods. For each method, we evaluated the result of its 2-minute run, using the result of its 10-hour run as the ground truth. The ground truth is the list of discovered context-subspace pairs during the 10-hour run, ranked by their exceptionality scores. Given the set of discovered context-subspace pairs in a 2-minute run, H, the coverage error is the average rank position of the pairs in the ground truth, defined by $Cov = \frac{1}{|H|} \sum_{(C,A) \in H} \text{rank}_{(C,A)}$.

Fig. 15 reports the average coverage error of each method under varying output size k. Table 4 shows the average and median coverage errors under varying beam width w. In Fig. 15, the coverage error of Beam-Conv is less than other methods by orders of magnitude, which suggests that Beam-Conv found highly-scored context-subspace pairs. Table 4 shows that coverage error decreases when beam width increases. The reason is that a wider beam leads to more patterns visited at every level and thus a better coverage of patterns. It is especially beneficial when highly-scored pairs reside in patterns at lower levels.

Table 4: The effect of beam width (w) on the coverage errors of top-10 context-subspace pairs of 10 entities. In each cell: the average and the median coverage errors. Both numbers are the smaller the better.

w	Beam-Rdm	Beam-Opt	Beam-Conv	Breath-First
3	3375.7/2636.9	2151.0/1071.5	**49.7/12.5**	383.0/390.5
4	3293.2/1607.3	2675.7/1622.1	**52.1/12.5**	383.0/390.5
5	2743.2/1871.2	2418.3/1550.8	**30.2/26.0**	383.0/390.5
6	2890.9/1809.2	2288.7/1259.7	**20.6/1.0**	383.0/390.5
7	2821.4/1398.9	1789.3/1259.7	**21.8/1.0**	383.0/390.5
8	2646.4/1818.6	1721.5/1168.8	**78.3/3.0**	383.0/390.5
9	2262.8/1653.4	1365.5/1107.3	**36.6/4.2**	383.0/390.5
10	2720.8/1619.9	1365.5/1107.3	**58.4/22.1**	383.0/390.5

7 RELATED WORK

In exceptional fact discovery, the output context-subspace pairs can be viewed as a way of explaining outliers. Most conventional outlier detection solutions, including those for graphs, focus on finding outliers but do not explain why they are outlying. For example, CODA [11] finds a list of community outliers, and FocusCO [19] clusters an attributed graph and then discovers outliers in the clusters. Besides the limitation that both approaches are only suitable for homogeneous graphs, it is up to users to figure out the explanations of the outliers. Although these two systems make such explanations easier by providing the communities or clusters in which the outliers reside, it still requires substantial expertise to summarize the communities/clusters' characteristics. A few works improve the interpretation of outliers' outlyingness [1, 16, 25]. For instance, work such as [1] and [16] use visualization to help users identify outliers and potentially discover their outlying aspects.

Although most existing outlying aspects mining approaches focus on finding global outlying aspects and do not consider contexts [10, 27], there are a few attempts to find contextual outlying aspects [2, 3, 24, 30]. The general framework Maverick allows users to adopt any exceptionality measure in the literature such as outlierness [3]. Although Maverick focuses on categorical attributes at this stage, it can be extended for numerical attributes so that measures such as skyline points [24], promotiveness [30], outlierness [2], outlyingness rank [10], and z-score [27] can be adopted in the framework.

8 CONCLUSION

In this paper, we study the problem of discovering exceptional facts about entities in knowledge graphs. Each exceptional fact consists of a pair (*context*, *subspace*). To tackle the challenge of exploring the exponential large search spaces of both contexts and subspaces, we propose a beam search based framework, Maverick, which applies a set of rules and heuristics during the discovery. The experiment results show that our proposed framework is both efficient and effective for discovering exceptional facts.

REFERENCES

[1] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. 2010. Oddball: Spotting anomalies in weighted graphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 410–421.

[2] Fabrizio Angiulli, Fabio Fassetti, Giuseppe Manco, and Luigi Palopoli. 2016. Outlying property detection with numerical attributes. *Data Mining and Knowledge Discovery* (2016), 1–30.

[3] Fabrizio Angiulli, Fabio Fassetti, and Luigi Palopoli. 2009. Detecting Outlying Properties of Exceptional Objects. *ACM Transactions on Database Systems* 34, 1, Article 7 (April 2009), 7:1–7:62 pages.

[4] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference*. 722–735.

[5] Roberto Bisiani. 1992. Beam Search. In *Encyclopedia of Artificial Intelligence* (2nd ed.), Stuart C Shapiro (Ed.). Wiley-Interscience, 1467–1468.

[6] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. 1247–1250.

[7] Mihaela A. Bornea, Julian Dolby, Anastasios Kementsietsidis, Kavitha Srinivas, Patrick Dantressangle, Octavian Udrea, and Bishwaranjan Bhattacharjee. 2013. Building an Efficient RDF Store over a Relational Database. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 121–132.

[8] Sarah Cohen, James T. Hamilton, and Fred Turner. 2011. Computational Journalism. *Commun. ACM* 54, 10 (Oct. 2011), 66–71.

[9] Sarah Cohen, Chengkai Li, Jun Yang, and Cong Yu. 2011. Computational Journalism: A Call to Arms to Database Researchers. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR)*. 148–151.

[10] Lei Duan, Guanting Tang, Jian Pei, James Bailey, Akiko Campbell, and Changjie Tang. 2015. Mining outlying aspects on numeric data. *Data Mining and Knowledge Discovery* 29, 5 (2015), 1116–1151.

[11] Jing Gao, Feng Liang, Wei Fan, Chi Wang, Yizhou Sun, and Jiawei Han. 2010. On community outliers and their efficient detection in information networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 813–822.

[12] Liqiang Geng and Howard J Hamilton. 2006. Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)* 38, 3 (2006), 9.

[13] Naeemul Hassan, Bill Adair, James T. Hamilton, Chengkai Li, Mark Tremayne, Jun Yang, and Cong Yu. 2015. The Quest to Automate Fact-Checking. In *Proceedings of the 2015 Computation+Journalism Symposium*.

[14] Naeemul Hassan, Afroza Sultana, You Wu, Gensheng Zhang, Chengkai Li, Jun Yang, and Cong Yu. 2014. Data In, Fact Out: Automated Monitoring of Facts by FactWatcher. *Proceedings of the VLDB Endowment (PVLDB), demonstration description* 7, 13 (2014), 1557–1560.

[15] Xiao Jiang, Chengkai Li, Ping Luo, Min Wang, and Yong Yu. 2011. Prominent streak discovery in sequence data. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1280–1288.

[16] U Kang, Jay-Yoon Lee, Danai Koutra, and Christos Faloutsos. 2014. Net-ray: Visualizing and mining billion-scale graphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 348–361.

[17] Edwin M Knox and Raymond T Ng. 1998. Algorithms for mining distancebased outliers in large datasets. In *Proceedings of the International Conference on Very Large Data Bases*. 392–403.

[18] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. 2010. Outlier detection techniques. In *Tutorial at the 16th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*.

[19] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. 2014. Focused Clustering and Outlier Detection in Large Attributed Graphs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[20] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient Algorithms for Mining Outliers from Large Data Sets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 427–438.

[21] Stuart Russell and Peter Norvig. 2010. *Artificial Intelligence: A Modern Approach (3rd Edition)* (3 ed.). Prentice Hall.

[22] Ron Rymon. 1992. Search through systematic set enumeration. *Technical Reports MS-CIS-92-66, Department of Computer and Information Science, University of Pennsylvania* (1992).

[23] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*. 697–706.

[24] Afroza Sultana, Naeemul Hassan, Chengkai Li, Jun Yang, and Cong Yu. 2014. Incremental Discovery of Prominent Situational Facts. In *Proceedings of the 30th International Conference on Data Engineering(ICDE)*. 112–123.

[25] Hanghang Tong and Ching-Yung Lin. 2012. Non-negative residual matrix factorization: problem definition, fast solutions, and applications. *Statistical Analysis and Data Mining* 5, 1 (2012), 3–15.

[26] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. 2009. Mining multi-label data. In *Data mining and knowledge discovery handbook*. Springer, 667–685.

[27] Nguyen Xuan Vinh, Jeffrey Chan, Simone Romano, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Jian Pei. 2016. Discovering outlying aspects in large datasets. *Data Mining and Knowledge Discovery* 30, 6 (Nov. 2016), 1520–1555.

[28] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[29] Xiang Wang and Ian Davidson. 2009. Discovering contexts and contextual outliers using random walks in graphs. In *Proceedings of the 9th IEEE International Conference on Data Mining*. 1034–1039.

[30] Tianyi Wu, Dong Xin, Qiaozhu Mei, and Jiawei Han. 2009. Promotion analysis in multi-dimensional space. *Proceedings of the VLDB Endowment* 2, 1 (2009), 109–120.

[31] You Wu, Pankaj K. Agarwal, Chengkai Li, Jun Yang, and Cong Yu. 2012. On "One of the Few" Objects. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1487–1495.

[32] You Wu, Pankaj K. Agarwal, Chengkai Li, Jun Yang, and Cong Yu. 2017. Computational Fact Checking through Query Perturbations. *ACM Transactions on Database Systems (TODS)* 42, 1 (Jan. 2017), 4:1–4:41.

[33] Yuehua Xu and Alan Fern. 2007. On learning linear ranking functions for beam search. In *Proceedings of the 24th International Conference on Machine Learning*. 1047–1054.

[34] Gensheng Zhang, Xiao Jiang, Ping Luo, Min Wang, and Chengkai Li. 2014. Discovering General Prominent Streaks in Sequence Data. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 8, 2 (June 2014), 9:1–9:37.

A APPENDIX

A.1 Optimization

A.1.1 Context defined by initial pattern P_0. Maverick starts with a trivial pattern P_0 in which the only node x_0 is a variable. According to Definition 2.2, every node in the knowledge graph G is a match to P_0. Therefore, the context defined by P_0 includes the nodes of all different types in G. Although Maverick allows such contexts consisting of heterogeneous entities, it brings two practical challenges. First, since different types of entities have different attributes, the exceptionality of an entity could be unrealistically bloated due to the sparsity of attributes that are common to many entities in the context. Second, the computation of exceptionality scores, of which the complexity is at least linear to the size of the context, is highly expensive when the context includes all the nodes in the data graph.

Due to these two reasons related to semantics and efficiency, it is more desirable that a context is homogeneous, i.e., it only includes entities of the same type. That imposes a requirement for a type system on the knowledge graph, which can be either predefined (e.g., DBpedia Ontology, Freebase Schema) or derived from the data graph. Maverick accommodates both kinds of type systems in knowledge graphs, which ensures its general applicability. Particularly, when an explicit type system does not exist, there can be different ways of deriving entity types. Although Maverick is oblivious to the specific approach for deriving the type system, the particular approach implemented in our system is as follows. In this approach, two entities u and v both belong to an implicit type if they have at least one common attribute, i.e., $A_u \cap A_v \neq \emptyset$ (see Definition 2.5 for A_v).[6] We then define the context $C_{v_0}^{P_0,x_0}$ for P_0 as:

$$C_{v_0}^{P_0,x_0} = \{u \in V_G | A_u \cap A_{v_0} \neq \emptyset\}.$$

[6] By this definition, a type is similar to a cluster of entities, and the clustering is non-exclusive, i.e., an entity can belong to multiple types.

For example, $C^{P_0, x_0}_{\text{G1}} = \{$G1, G2, G3, G4, G5$\}$, which excludes team nodes (such as CRO), player nodes (such as S1), and so on. Note that this way of deriving entity types is compatible with Freebase's type system, in which the types of the source/destination nodes of an edge are determined by the label (i.e., type) of the edge.

A.1.2 Subspace pruning across contexts. Since Maverick only needs to return top-k context-subspace pairs, the Exceptionality Evaluator (Alg. 3) employs a simple optimization that uses the current top-k context-subspace pairs H to prune the subspaces in a new context. Specifically, a subspace in a context is pruned if its upper bound exceptionality score is lower than the score of the k-th best context-subspace pair in H. This optimization is simply implemented by modifying the function EXCEPTIONALITY-EVALUATOR in Alg. 3 to accept an additional argument H and changing Line 2 from `Tk` $\leftarrow \emptyset$ to `Tk` $\leftarrow H$. The experiment results reported in Section 4 already reflect this optimization.

A.2 Case Study

To illustrate the effectiveness of Maverick, we present below some examples of exceptional facts discovered by Maverick in both graph WCGoals and graph OscarWinners.

Goal(46683) is the only own goal in Brazil's World Cup history.

Exceptionality $\chi_o = 0.986$
Subspace $\{(\textit{awared-to}, \rightarrow)\}$
Context $C^{P, x_0}_{\text{Goal(46683)}}$, where $P = \{(x_0, \textit{scored-by}, x_1), (x_1, \textit{play-for}, \text{BRA})\}$

Indeed, among all the 221 goals that were scored by Brazil players in the *FIFA World Cup Finals* tournaments, Goal(46683), which was awarded to Croatia, was the only goal not awarded to Brazil. This exceptional fact has a very high score.

Among all the crew members of Oscar winning films, Paul J. Franklin (FilmCrew(7674)) is the only crew member with role Computer Animation.

Exceptionality $\chi_f = 0.784$
Subspace $\{(\textit{film-crew-role}, \rightarrow)\}$
Context $C^{P_0, x_0}_{\text{FilmCrew(7674)}}$

This example demonstrates the utility of Maverick in revealing data errors, as motivated in Section 1. While investigating why this entity is exceptional, an analyst will realize the exceptional fact is due to a data error. An edge mistakenly links from node Paul J. Franklin to a genre node Computer Animation which is incorrectly used in this case as a role node. The correct crew role node should have been Computer Animator.

Goal(23464) is the only goal awarded to Paraguay, among all the goals scored in matches hosted in Mexico City that had at least two goals.

Exceptionality $\chi_f = 0.983$
Subspace $\{(\textit{awared-to}, \rightarrow)\}$
Context $C^{P, x_1}_{\text{Goal(24227)}}$, where $P = \{(x_0, \textit{goal}, x_1), (x_0, \textit{goal}, x_2), (x_0, \textit{venue}, \text{Mexico City})\}$

There are in total 62 goals scored in matches hosted in Mexico City, among which 58 were scored in 18 multiple-goal matches. These 58 goals were awarded to 12 different teams. Paraguay is the only team that was awarded only one of the 58 goals.

Game(899) is one of the only two games in which the home team was Senegal, among all the games where there was a player wearing the number 21 shirt.

Exceptionality $\chi_f = 0.959$
Subspace $\{(\textit{home}, \rightarrow)\}$
Context $C^{P, x_1}_{\text{Game(899)}}$, where $P = \{(x_0, \textit{bibnum}, \text{Bibnum(21)}), (x_0, \textit{participate-in}, x_1)\}$

In 761 games some player wore number 21. Game(899) is one of the only two such games in which the home team was Senegal.

Among the Oscar winning films produced in the United States, The Lord of the Rings: The Return of the King (Film(31768)) is one of the only 7 films that were also produced in New Zealand.

Exceptionality $\chi_o = 0.676$
Subspace $\{(\textit{country}, \rightarrow)\}$
Context $C^{P, x_0}_{\text{Film(31768)}}$, where $P = \{(x_0, \textit{country}, \text{USA})\}$

There are in total 662 Oscar winning films produced in the United States, of which 545 were produced solely in the United States. Only 7 of the co-produced films were co-produced in New Zealand. However, the score of this fact is not as high as that of the last two facts, because China, Brazil, and a few other countries co-produced even less films.

A.3 Proof of Theorem 4.4

PROOF. The tightest upper bound of δ is the exact score itself, validating the above upper bound $\delta_{\max} = \delta(v, A, C)$. With regard to δ_{min}, let w be the nearest neighbor of v with respect to A, then $\delta(v, A, C) = -d(v, w, A)$ according to Eq. (5). Based on Eq. (6), $d(v, v', A) \leq d(v, v', A') \leq d(v, v', A_v)$ for any v' and $A \subseteq A' \subseteq A_v$. Hence,

$$\begin{aligned}\delta(v, A', C) &\geq -d(v, w, A_v) = -\sqrt{d(v, w, A)^2 + d(v, w, A_v \setminus A)^2} \\ &\geq -\sqrt{d(v, w, A)^2 + |A_v| - |A|} \\ &= -\sqrt{\delta(v, A, C)^2 + |A_v| - |A|} = \delta_{\min}(v, A, C)\end{aligned}$$

■