CD-004

出現分布に基づく知識グラフの例外検出
Outlier Detection Based on Occurence Distribution in Knowledge Graph

加藤 遼[1)]　堀内 美聡[1)]　松田 光司[1)]　佐々木 勇和[1)]　鬼塚 真[1)]
Ryo Kato　Misato Horiuchi　Koji Matsuda　Yuya Sasaki　Makoto Onizuka

あらまし　知識グラフの例外検出は，知識グラフから通常と異なる傾向の事柄を例外として検出する技術であり，誤り検出や知識抽出に役立てることができる．既存研究は，検出する例外の種類が少ない，および出現分布の特性を反映しないという課題がある．本研究では，多様な例外を検出する例外検出手法を提案する．提案手法では，知識グラフのサブグラフであるパターンにあてはまる各エンティティの隣接要素に基づいて，隣接要素の出現分布の特性を反映可能な例外スコアを定義する．さらに，パターンの探索効率化のため，例外スコアの上界によりパターンの枝刈りを行う．評価実験において，枝刈りにより検出の効率性と例外的事実の有用性を評価する．

1　はじめに

知識グラフはさまざまな知識のつながりをグラフ構造で表したものである．この知識は，主語・述語・目的語の文型で表される．名詞的な性質を持つエンティティ同士が，述語的な性質を持つ有向ラベル付きリンクによって接続されている．知識グラフは，情報の構造化により計算機が利用しやすく知見を得やすいため，問合せ [1] や Web 検索 [2] において活用されている．知識グラフ上には，よく見られる傾向と異なる傾向の事柄が含まれることがある．そのような事柄は，例外スコアと呼ばれる指標を用いて例外的な事柄として検出することができる [3]．得られた例外的な事柄は，誤って登録されたデータや特徴的な価値のある情報であることが多いため，誤り検出や知識抽出に活用することができる．

知識グラフにおける例外検出手法として Maverick [3] や FMiner [4] が提案されている．これらは，コンテキストと呼ばれる同じサブグラフにマッチするエンティティを比較対象に例外検出を行う手法である．しかし，これらには 3 つの課題が存在する．一つ目に，検出可能な例外の種類が少ない．これは， Maverick はエンティティ，FMiner はエンティティラベルの相違のみを対象として例外検出を行なっており，リンクの相違を例外として出力することができないためである．二つ目に，ある 1 つのエンティティの数が極端に少ないといった，エンティティの出現分布の特性を反映できない．これは，いずれの手法も確率ベースで例外スコアを計算しているためである．三つ目に，複数のエンティティに着目して例外検出を行いたい場合，そのエンティティ数と同じ回数だけ実行する必要がある．これは着目するエンティティを事前に決定する必要があるためである．また，同時に複数のエンティティに着目して例外検出を行う場合，知識グラフ上に存在する全てのコンテキストを探索する方法では膨大な時間がかかる．これは，知識グラフ中には多数のエンティティ，リンク，エンティティのラベルが存在するためである．

そこで本稿では，隣接する様々な要素をその出現分布に基づいて検出する知識グラフの新たな例外検出フレームワークを提案する．多様な例外の検出と隣接要素の出現分布を反映するような新たな例外スコアを提案し，全てのエンティティを対象にする例外検出を高速化を用いながら行う．知識グラフ上の特定のコンテキストにあてはまるエンティティと比較し，隣接要素（エンティティラベル，固有エンティティ，リンク）の相違，欠落，余分に着目することで，多くの種類の例外を検出する．例外スコアは，隣接要素の出現分布の偏りに基づいた計算法により，隣接要素の出現分布の特性を反映する．提案する例外検出手法では，一回の実行で全てのエンティティの隣接する要素を対象として例外検出を行う．各コンテキストを全て探索する方法では時間がかかるため，あるコンテキストにあてはまるエンティティがとりうる例外スコアの最大値である上界スコアを用い， top-k に入らなければそのコンテキストを枝刈りする．この上界スコアは，隣接要素の分布から算出される．この上界スコアの利用により効率的な探索が可能となる．

実験では，知識グラフ NELL [5] および DBpedia [6] を用いて，検出効率と例外的事実の有用性の評価を行う．検出効率の評価実験では，リンク数を変化させながら，素朴な手法と枝刈りを用いた提案手法での実行時間を比較し，提案手法が効率的に例外検出ができることを示す．例外的事実の有用性の評価では，出力された例外的事実の内容の分析により利用例を考察し，出力された例外的事実が有用であることを示す．

本稿の構成は次の通りである．まず，2 章で関連研究を示し，3 章にて知識グラフ等の事前知識の説明と問題定義を行う．4 章に例外スコアの計算方法を，5 章にてフレークワークを示す．6 章にて実験結果を示し，7 章にて本稿をまとめる．

2　関連研究

本章では，知識グラフの例外検出手法を紹介する．その後，知識グラフ以外のグラフの例外検出の既存研究を紹介する．

Maverick [3] は，ユーザによって入力される一つのエンティティを，それと同じサブグラフにマッチする他のエンティティと比較する．そして，あるリンクの行先が例外的に異なる場合，その行先を例外として出力する．さらに明確な例外理由として，比較対象としたサブグラフやリンクも出力する．Maverick の例外スコアはユーザが自由に定義でき，その一例として隣接エンティティのカウント数に関する確率ベースの手法が提案されている．FMiner [4] は， Maverick において，マッチさせるサブグラフをパス状のものに限定し，入力したエンティティのラベル自体を例外の対象として例外検出を行う手法である．しかしこれらには，エンティティラベル，エンティティ，リンクといった多様な例外を検出できない，出現分布の特性を反映できない，例外対象とするエンティティを指定する必要があるという課題がある．

1）　大阪大学大学院 情報科学研究科

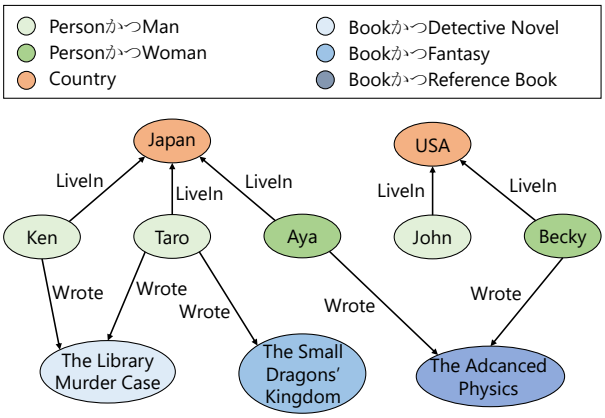


図 1: 知識グラフの例

グラフを対象とした例外検出技術は様々なものが提案されている．しかし，知識グラフには付加情報であるノードやエッジのラベルがあるため，以下で述べる技術はそのまま知識グラフに適用することはできない．属性付きグラフにおける例外検出として， FocusCO [7] が提案されている． FocusCO は，属性に対する重み付けの学習により，構造的に同質ではあるが属性的に大きく異なるノードを例外として検出する．重み付きグラフにおける例外検出として， OddBall [8] や Net-ray [9] が提案されている．OddBall [8] では，次数や隣接リンクの重み和等の近傍に関する特徴量の間に冪乗則が見られることを利用して，近傍が異常であるようなノードを検出する手法を提案している．Net-ray [9] では，規模の大きな重み付きグラフの隣接行列の可視化がメインのタスクとして提案されている．その隣接行列のプロットに対してクラスタリングを行い，シングルトンクラスタを見つけることによって，例外的なリンクを見つける．他に，重み・ラベルなしグラフにおける例外検出として， CODA [10] や Gupta ら [11] の手法が研究されている．CODA では，各ノードの近傍ノードの情報を基にコミュニティの割当確率を定義し，コミュニティ割当による事後確率の最大化により，コミュニティ内での例外検出を行う．Gupta らは，最大マージンの考えに基づき，クエリのコミュニティに属するノードのうち近傍が例外であるエンティティを検出する手法を提案している．

3 事前定義と問題定義

本章では，知識グラフ，パターン，コンテキスト，例外スコアに関する定義を行なった後，知識グラフの例外検出の問題定義を行う．

定義 1 (知識グラフ)知識グラフを，$G = (\mathcal{V}, \mathcal{T}, \mathcal{L}_V, \mathcal{L}_R)$ とする．$\mathcal{V}$ はエンティティ集合，$\mathcal{T} \subseteq \mathcal{V} \times \mathcal{L}_R \times \mathcal{V}$ はトリプル集合，$\mathcal{L}_V$ はエンティティラベル集合，$\mathcal{L}_R$ はリレーションタイプ集合を表す．各エンティティ $v \in \mathcal{V}$ には，$L(v) \subseteq \mathcal{L}_V$ となるようなラベル集合 $L(v) \neq \phi$ が存在する．

図 1 の知識グラフでは，$\mathcal{V}$ は Japan, Ken 等の計 10 個のエンティティで構成され，$\mathcal{T}$ は (Ken, LiveIn, Japan), (Becky, Wrote, The Advanced Physics) 等の計 10 個のトリプルで構成される．$\mathcal{L}_V$ = {People, Man, Woman, Country, Book, Detective Novel, Fantasy, Reference Book}，$\mathcal{L}_R$ = {LiveIn, Wrote} となる．各エンティティの持つラベル集合は，例として L_V(Ken) = {People, Man} となる．

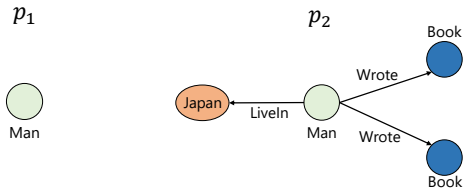


図 2: $|\mathcal{T}_p| = 0$ のパターン例 図 3: $|\mathcal{T}_p| = 3$ のパターン例

定義 2 (有向リレーションタイプ)$L_R^{(D)} = \mathcal{L}_R \cup \{r^{-1} \mid r \in \mathcal{L}_R\}$ を有向リレーションタイプとする．

図 1 の知識グラフでは，$L_R^{(D)}$ = {LiveIn, Wrote, LiveIn^{-1}, Wrote^{-1}} となる．

定義 3 (パターンとパターンエンティティ)パターンを $p = (\mathcal{V}_p, \mathcal{T}_p)$ とする．$\mathcal{V}_p \subseteq \mathcal{L}_V \cup \mathcal{V}$ はパターンエンティティ集合，$\mathcal{T}_p \subseteq \mathcal{V}_p \times \mathcal{L}_R \times \mathcal{V}_p$ はパタートリプル集合を表す．

図 2 のパターン p_1 では，$\mathcal{V}_{p_1}$ = {Man}, $\mathcal{T}_{p_1} = \phi$ となる．図 3 のパターン p_2 では，$\mathcal{V}_{p_2}$ = {Man, Japan, Book, Book} となり，$\mathcal{T}_{p_2}$ は (Man, LiveIn, Japan) 等の計 4 つのパタートリプルによって構成される．

定義 4 (コンテキスト)コンテキストを $C = (v_c, p)$ とする．p はパターン，v_c は p 中のパターンエンティティを表す．

図 3 のパターン p_2 と p_2 中のパターンエンティティ Man から成るコンテキスト C_1 = (Man, p_2) は，「ある本を 2 冊書いた，日本に住んでいる男性」を意味する．

定義 5 (コンテキストへのマッチ)コンテキスト $C = (v_c, p)$ と知識グラフ G が与えられた際に，全ての $v_p, v'_p \in \mathcal{V}_p$ について，$v_p \in \mathcal{L}_V$ のとき $v_p \in L\left(h(v_p)\right)$，$v_p \in \mathcal{V}$ のとき $h(v_p) = v$，および，$(v_p, r, v'_p) \in \mathcal{T}_P$ であれば $\left(h(v_p), r, h(v'_p)\right) \in \mathcal{T}$ となる $\mathcal{V}_p \to \mathcal{V}$ のマッピング関数 h に対して，C にマッチするエンティティの集合 $\mathcal{M}(C, G) \subseteq \mathcal{V}$ は，$v_c \in \mathcal{L}_V$ のとき $v_c \in L(v)$，$v_c \in \mathcal{V}$ のとき $h(v_c) = v$ となるような全てのエンティティ $v \in \mathcal{V}$ の集合である．

図 1 の知識グラフ G と，図 2 のパターン p_1 と p_1 中のパターンエンティティ Man から成るコンテキスト C_1 = (Man, p_1) において，$\mathcal{M}(C_1, G)$ = {Ken, Taro, John} となる．図 3 のパターン p_2 と p_2 におけるパターンエンティティ Man から成るコンテキスト C_2 = (Man, p_2) において，$\mathcal{M}(C_2, G)$ = {Taro} となる．

定義 6 (例外スコア)知識グラフ G において，コンテキスト C におけるトリプル t の例外さを示すスコアを $S(t, C, G)$ で定義する．コンテキスト C におけるエンティティ v の例外さを示すスコアを $S(v, C, G)$ で定義する．

知識グラフの例外検出の問題定義　以下の入出力によって，例外スコアが top-k となるような知識グラフ中のトリプルまたはエンティティを出力する．

- 入力：知識グラフ G，出力するトリプルまたはエンティティの数 k，パターンの最大トリプル数 N
- 出力：top-k の例外スコアを持つトリプルまたはエンティティ，同じコンテキストに属するエンティティの各隣接要素の出現分布とそのコンテキスト

なお，コンテキストおよびそのコンテキストに属するエンティティの各隣接要素の出現分布は，例外理由として解釈することができる．

4　例外スコア

本章では，例外の種類について述べた後，例外スコアの計算方法を述べる．

4.1　例外の種類についての定義

コンテキスト C において，エンティティ $v \in \mathcal{M}(C,G)$ の隣接要素を $\mathcal{M}(C,G)$ 中の他のエンティティの各隣接要素と比較することで，例外さを評価する．これにより，相違における例外，欠落における例外，余分における例外を検出する．相違における例外は，隣接するエンティティラベル，エンティティ，リンクが例外的に異なっていることを示す例外である．欠落における例外は，エンティティ，リンクが例外的に欠落していることを示す例外である．余分における例外は，エンティティ，リンクが例外的に余分であることを示す例外である．それぞれの例外について具体的に説明する．

同じコンテキストにマッチするエンティティと接続するエンティティおよびエンティティラベルの例外さを評価するために，v と $\mathcal{M}(C,G)$ 中の他のエンティティそれぞれの有向リレーションタイプ $r^{(D)}$ によって結合しているエンティティを比較する．このとき，v と結合しているエンティティ v' のラベル（エンティティ v'）が $\mathcal{M}(C,G)$ 中の他の各エンティティと結合しているエンティティのラベル（エンティティ）と比較して数が少ないとする．このとき，$r^{(D)} \in \mathcal{L}_R$ の場合トリプル $t = (v, r^{(D)}, v')$，$r^{(D)} \in \{r^{-1} \mid r \in \mathcal{L}_R\}$ の場合トリプル $t = (v', (r^{(D)})^{-1}, v)$ はエンティティラベル（エンティティ）における例外的なトリプルである．また，$\mathcal{M}(C,G)$ に属するエンティティは $r^{(D)}$ と接続している（接続していない）ものが多いが，v は接続していない（接続している）とき，エンティティ v は，欠落（余分）エンティティにおける例外的なエンティティである．

同じコンテキストにマッチするエンティティと接続する有向リレーションタイプの例外さを評価するために，v と $\mathcal{M}(C,G)$ 中の他のエンティティそれぞれとラベル l のエンティティの間にある有向リレーションタイプ $r^{(D)}$ を比較する．このとき，v と結合している有向リレーションタイプ $r^{(D)}$ が $\mathcal{M}(C,G)$ 中の他の各エンティティと結合している有向リレーションタイプと比較して数が少ないとする．このとき，同様にトリプル $t = (v, r^{(D)}, v')$ または $(v', (r^{(D)})^{-1}, v)$ はリレーションにおける例外的なトリプルである．また，$\mathcal{M}(C,G)$ に属するエンティティはラベル l のエンティティと接続している（接続していない）ものが多いが，v は接続していない（接続している）とき，エンティティ v は，欠落（余分）リレーションにおける例外的なエンティティである．

4.2　例外スコアの計算方法

本節では，トリプルやエンティティの例外さを示す例外スコアについて説明する．コンテキスト C 中でのトリプル t の例外スコア $S(t,C,G)$ は (1) 式のように計算される．この例外スコアは，相違における例外と余分における例外の評価に使用される．なお，t の 2 つのエンティティのうち，C にマッチするエンティティが v，もう一方のエンティティが v' である．t のリレーションタイプを r としたとき，v が始点エンティティのときは $r^{(D)} = r$，終点エンティティのときは $r^{(D)} = r^{-1}$ である．

$$S(t,C,G) = \max\{\max_{l \in L(v')} \left\{S_L(v, r^{(D)}, l, C, G)\right\}, S_U(v, r^{(D)}, v', C, G), \max_{l \in L(v')} \left\{S_R(v, r^{(D)}, l, C, G)\right\}\} \quad (1)$$

(1) 式の各項の詳細な導出方法は後述する．

続いて，コンテキスト C 中でのエンティティ v の例外スコア $S(v,C,G)$ は (2) 式のように計算される．この例外スコアは，欠落における例外の評価に使用される．なお，v は C にマッチするエンティティとなっている．

$$S(v,C,G) = \max\{S'_L(v,C,G), S'_U(v,C,G), S'_R(v,C,G)\} \quad (2)$$

これらの各項は以下のように計算される．

$$S'_L(v,C,G) = \max_{r^{(D)} \in L_R^{(\varepsilon)}(v,C)} S_L(v, r^{(D)}, \phi, C, G) \quad (3)$$

$$S'_U(v,C,G) = \max_{r^{(D)} \in L_R^{(\varepsilon)}(v,C)} S_U(v, r^{(D)}, \phi, C, G) \quad (4)$$

$$S'_L(v,C,G) = \max_{l \in L_V^{(\varepsilon)}(v,C)} S_R(v, \phi, l, C, G) \quad (5)$$

ここで，$L_R^{(\varepsilon)}(v,C)$ は，$\mathcal{M}(C,G)$ のいずれかのエンティティと結合しているが v とは結合していない有向リレーションタイプの集合である．$L_V^{(\varepsilon)}(v,C)$ は，$\mathcal{M}(C,G)$ のいずれかのエンティティと 1 ホップ先で結合しているが v とは結合していないエンティティラベルの集合である．また，(3) 式，(4) 式，(5) 式で使用されている関数 $S_L(v, r^{(D)}, \phi, C, G), S_U(v, r^{(D)}, \phi, C, G), S_R(v, \phi, l, C, G)$ は，全て (1) 式中の関数と同じ関数である．

エンティティラベルにおける例外スコア $S_L(v, r^{(D)}, l, C, G)$ は，v から有向リレーションタイプ $r^{(D)}$ によって結合しているエンティティラベル l と，$\mathcal{M}(C,G)$ の各エンティティから $r^{(D)}$ によって結合しているエンティティラベルを比較したときに，l が例外的に少ないほど高くなるようなスコアである．なお，$l = \phi$ とすれば，「v は $r^{(D)}$ よって結合しているエンティティがない」と捉えることができ，欠落エンティティにおける例外の評価に拡張することができる．

計算方法としては，まず，$\mathcal{M}(C,G)$ の各エンティティから $r^{(D)}$ によって結合しているエンティティの数を，そのエンティティラベル毎にカウントし，それらを $a_1, ..., a_{n_a}$ とする．そして，$\mathcal{M}(C,G)$ のうち $r^{(D)}$ によっ

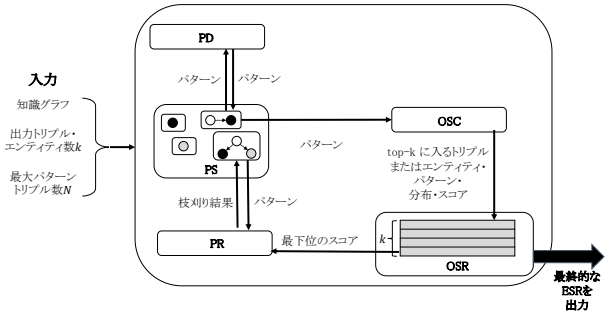


図 4: 知識グラフの例外検出フレームワーク

て結合していないエンティティの数をカウントし，その値を a_ϕ とする．そして，$l \neq \phi$ の場合は $a_1, ..., a_{n_a}$ のうちエンティティラベル l のカウント数であるものを a_k とし，$l = \phi$ の場合は $a_k = a_\phi$ とする．そして，以下の式によって $S_L(v, r^{(D)}, l, C, G)$ を計算する．

$$S_L(v, r^{(D)}, l, C, G) = \frac{1}{N_a} \left\{ \sum_{i=1}^{n_a} \left(\frac{a_i}{a_k} \right)^2 + \left(\frac{a_\phi}{a_k} \right)^2 \right\} \quad (6)$$

$$N_a = \begin{cases} n_a + 1 & (a_\phi \neq 0) \\ n_a & (a_\phi = 0) \end{cases} \quad (7)$$

同様の手法で，固有エンティティにおける例外スコア $S_U(v, r^{(D)}, v', C, G)$，リレーションにおける例外スコア $S_R(v, r^{(D)}, l, C, G)$ を求めることができる．ただし，エンティティラベルにおける例外スコアと異なり，$S_U(v, r^{(D)}, v', C, G)$ は，v と $r^{(D)}$ によって結合している v' の例外さを示すスコアであり，$S_R(v, r^{(D)}, l, C, G)$ は v とラベル l のエンティティとの間の $r^{(D)}$ の例外さを示すスコアである．

まず，$S_U(v, r^{(D)}, v', C, G)$ は，(6) 式の $a_1, ..., a_\phi$ をエンティティ毎にその数をカウントしたものとし，a_k を $a_1, ..., a_\phi$ のうち v' のカウント数であるものとすることによって計算することができる．次に，$S_R(v, r^{(D)}, l, C, G)$ は，(6) 式の $a_1, ..., a_\phi$ をラベル l のエンティティとの間の有向リレーションタイプ毎にその数をカウントしたものとし，a_k を，$a_1, ..., a_\phi$ のうち $r^{(D)}$ のカウント数であるものとすることによって計算することができる．

5 フレームワーク

本章では，知識グラフにおける例外検出フレークワークについて述べる．

5.1 フレームワークの概要

図 4 に例外検出フレームワークを示す．例外検出フレームワークは，パターン発見器 (PD)，枝刈り器 (PR)，例外スコア計算器 (OSC)，パターンストア（PS），例外スコアランキング (OSR) の 5 つの要素によって構成される．パターン発見器では，知識グラフ中に存在するパターンを発見する．枝刈り器では，入力パターンに対し，5.2 節にて後述する枝刈りを行う．例外スコア計算器では，入力コンテキストにマッチする全てのエンティティまたはそれを含む全てのトリプルの例外スコアを計算する．パターンストアでは，パターン発見器によって発見されたパターンを格納する．例外スコアランキングでは，上位 k 個の例外スコアのエンティティまたはトリプルと，その例外スコアを算出する際に用いたコンテキストおよび隣接要素の出現分布を格納する．

例外スコアの計算の際は，パターンストア中のパターン p とパターンエンティティ $v_c \in \mathcal{V}_p$ から成るコンテキスト $C = (v_c, p)$ を入力として，C にマッチする全てのエンティティ v または v を含む全てのトリプル t の例外スコア $S(v, C, G), S(t, C, G)$ を例外スコア計算器で計算する．そのスコアが top-k である場合，例外スコアランキングに，v または t，C，隣接要素の出現分布を格納する．枝刈りの際は，パターンストア中のパターンを入力として，パターンに対して枝刈りを行う．そして，その枝刈り結果を入力元のパターンに返す．

Algorithm 1: 上界スコア $U^{(L)}(C, G)$

input : context C, knowledge graph G
output : the upper bound score which an entity $v \in \mathcal{M}(C, G)$ or a triple which has an entity v can calculate

$L_R^{(\chi)}(C, G) \leftarrow \phi$
$U_{max} \leftarrow 0$
for $v \in \mathcal{M}(C, G)$ **do**
  $L_R^{(\chi)}(v) \leftarrow$ all directed relation types who are combined with v
  $L_R^{(\chi)}(C, G) \leftarrow L_R^{(\chi)}(C, G) \cup L_R^{(\chi)}(v)$
for $r^{(D)} \in L_R^{(\chi)}(C, G)$ **do**
  $U_1 \leftarrow 1, U_2 \leftarrow 1$
  $A(r^{(D)}) \leftarrow \{a_i \mid i = 1, ..., n_a\}$
  $A' \leftarrow \{1\}$
  if $a_\phi \neq 0$ **then**
    $A(r^{(D)}) \leftarrow A(r^{(D)}) \cup \{a_\phi\}$
  Sort $A(r^{(D)})$ in descending order Pop the minimum of $A(r^{(D)})$
  for $a \in A(r^{(D)})$ **do**
    $A' \leftarrow A' \cup a$
    $U_2 \leftarrow \frac{1}{|A'|} \sum_{a' \in A'} a'^2$
    if $U_2 < U_1$ **then**
      break
    $U_1 \leftarrow U_2$
  $U_{max} \leftarrow \max(U_{max}, U_1)$
return U_{max}

5.2 枝刈り器

知識グラフ中にはエンティティ，エンティティラベル，リンクラベルが多数存在し，知識グラフ上の全てのコンテキストを探索する全探索ではコストがかかる．そこで，近似なしで高速化を行う枝刈り手法を提案する．提案する枝刈り手法では，あるコンテキストを入力とする例外スコアの最大値（以下，上界スコア）を用いて，明らかに top-k の例外スコアを算出しないようなコンテキストを入力とする例外スコアの計算を省略する．また，段階的な枝刈りによって，省略可能な処理をできるだけ省略し，無駄のない枝刈りを行う．

5.2.1 上界スコア

本節では，あるコンテキスト C から算出される，エンティティラベルにおける上界スコア $U^{(L)}(C, G)$，固有エンティティにおける上界スコア $U^{(U)}(C, G)$，リレーションにおける上界スコア $U^{(R)}(C, G)$ の計算方法について説明する．まず，$U^{(L)}(C, G)$ の計算方法について述べる．上界スコアは，パターン p にパターントリプルを 1 つ増やしたパターンを $p^{(new)}$ としたときに，任意のパターンエンティティ $v_p \in \mathcal{V}_p$ において

$\mathcal{M}((v_p, p^{(new)}), G) \subseteq \mathcal{M}((v_p, p), G)$ が成立することを利用して (8) 式のように計算される．

$$U^{(L)}(C,G) = \max_{a'_k \in \{a'_1,...,a'_{n_a},a'_\phi\}} \max_{a'_1 \in [0,a_1]} \max_{a'_2 \in [0,a_2]} ... \max_{a'_n \in [0,a_n]} \max_{a'_\phi \in [0,a_\phi]} \frac{1}{N'_a} \left\{ \sum_{i=1}^{n_a} \left(\frac{a'_i}{a'_k}\right)^2 + \left(\frac{a'_\phi}{a'_k}\right)^2 \right\} \quad (8)$$

$$N'_a = |\{i \mid a'_i \neq 0, i = 1, ..., n_a, \phi\}| \quad (9)$$

しかし，(8) 式を直接計算するのはコストが大きいため，効率化する．ある 1 種類の隣接要素のカウント数を 1 とし，どの隣接要素のカウント数が 0 となったときに例外スコアが最大になるかという方法により行う．

$U^L(C,G)$ を計算するアルゴリズムを Algorithm 1 に示す．まず，$\mathcal{M}(C,G)$ のいずれかのエンティティと結合している全ての有向リレーションタイプを $L_R^{(\chi)}(C,G)$ とする（3–5 行目）．そして，この $L_R^{(\chi)}(C,G)$ を順に走査する（6–19 行目）．各有向リレーションタイプのループ内では，以下の方法により (8) 式と同じ値 $U^{(L)}_{r^{(D)}}(C,G)$ を求める．そして，$r^{(D)} \in L_R^{(\chi)}(C,G)$ についての $U^{(L)}_{r^{(D)}}(C,G)$ の最大値を $U^{(L)}(C,G)$ とする．

まず，初期状態として，$a_1, ..., a_{n_a}, a_\phi$ のうち最小値を 1，それ以外を 0 とし，整数 n を 2 とする．次に，元の $a_1, ..., a_{n_a}, a_\phi$ のうち，最大値から $n-1$ 番目に大きい値を元の値に戻し，$a_k = 1$ として (6) 式の右辺を計算して得られる値を $U(n)$ とする．このとき，n は (6) 式の右辺中の N_a と一致する．$U(n) > U(n-1)$ かつ $n \neq n_a + 1$ の場合は，n をインクリメントし同様の処理を行う．なお，$U(1) = 1$ とする．一方，そうでない場合は，$n \neq n_a + 1$ ならば $U^{(L)}_{r^{(D)}}(C,G) = U(n-1)$ とし，$n = n_a + 1$ ならば $U^{(L)}_{r^{(D)}}(C,G) = U(n)$ とする．

この求め方が正しい証明を以下に示す．まず，(8) 式において，a'_k として $\{a'_1, ..., a'_{n_a}, a'_\phi\}$ のうちのどれを選択すれば最大になるかを考える．これは，max の項の中の分母が最小となるため，$a'_k = \min\{a'_1, ..., a'_{n_a}, a'_\phi\}$ を選択すると最大となる．次に，a'_k の値が何であれば最大になるかを考える．これは，max の項の中の分母が最小となるため，$a'_k = 1$ のときに最大となる．次に，$\{a'_1, ..., a'_{n_a}, a'_\phi\} \backslash a'_k$ の各値が何であれば最大になるかを考える．これは，$a'_i \neq 0$ であれば，max の項の中の分子が最大となるため，$a'_i = a_i$ のときに最大となる．一方，$a'_i = 0$ であれば，N'_a が小さくなることにより例外スコアが大きな値をとるため，$a'_i = 0$ のときに最大となる可能性もある．次に，$\{a'_1, ..., a'_{n_a}, a'_\phi\} \backslash a'_k$ のうち 0 をとらない値が M 個であるときにどの値が 0 でなければ最大になるかを考える．これは，より大きな値が 0 でないほど例外スコアが大きな値をとるため，$\{a'_1, ..., a'_{n_a}, a'_\phi\} \backslash a'_k$ の top-M が 0 でないときに最大となる．

以上の議論により，$U^{(L)}(C,G)$ は以下のように書き換えることができる．なお，式中の A'_M は，$\{a_1, ..., a_{n_a}, a_\phi\} \backslash a_k$ の top-M 集合である．

$$U^{(L)}(C,G) = \max_{M \in [0,n_a]} \frac{1}{M+1} \left(\sum_{a' \in A'_M} a'^2 + 1 \right) \quad (10)$$

次に，M を 0 から順に 1 ずつ増やし，$M = m$ のときの max の項の中の値 $U(m+1)$ を計算することによって (10) 式の値を求める手順を想定する．ある $M = m$ において初めて $U(m+1) < U(m)$ となったとき，$\{a'_1, ..., a'_{n_a}, a'_\phi\} \backslash a'_k$ のうち m 番目に大きい値 α について $\alpha^2 < U(m+1) < U(m)$ という関係が成り立つ．$M = m+1$ 以降では，α 以下の値のみを平均として加えるため，$U(m+2), ..., U(n_a+1) < U(m+1)$ が成立する．また，$M = m$ において初めて $U(m+1) < U(m)$ となるため，$U(0) < ... < U(m)$ が成立する．以上のことから，$U(m)$ が (10) 式と等しくなり，先述の求め方が正しいことが証明された．

$U^{(U)}(C,G), U^{(R)}(C,G)$ においても， Algorithm 1 に以下の変更を加えることで計算することができる．まず，$U^{(U)}(C,G)$ の場合は，8, 10, 11 行目の $a_1, ..., a_{n_a}, a_\phi$ を，$S_U(v, r^{(D)}, v', C, G)$ の計算の際に用いたカウント数とすることで計算することができる．次に，$U^{(R)}(C,G)$ の場合は，以下の変更によって計算することができる．

- 4 行目の $L_R^{(\chi)}(v)$ を，v の 1 ホップ先に存在する全てのエンティティラベル $L_V^{(\chi)}(v)$ とし，5 行目以降の $L_R^{(\chi)}(C,G)$ が，$\mathcal{M}(C,G)$ のいずれかのエンティティの 1 ホップ先に存在する全てのエンティティラベル集合 $L_V^{(\chi)}(C,G)$ となるようにする．
- 6 行目における走査対象を，各エンティティラベル $l \in L_V^{(\chi)}(C,G)$ とする．
- 8 行目以降の $A(r^{(D)})$ を，$C(l)$ とする．
- 8, 10, 11 行目の $a_1, ..., a_{n_a}, a_\phi$ を，$S_R(v, r^{(D)}, l, C, G)$ の計算の際に用いたカウント数にする．

5.2.2 枝刈りの種類

枝刈りには，例外の種類レベルでの枝刈り，パターンエンティティレベルでの枝刈り，パターンレベルでの枝刈りの 3 種類がある．パターン p，パターンエンティティ v_c において，エンティティラベルにおける例外の種類レベルでの枝刈り可能性を $\theta^{(L)}(v_c, p) \in \{0,1\}$，固有エンティティにおける例外の種類レベルでの枝刈り可能性を $\theta^{(U)}(v_c, p) \in \{0,1\}$，リレーションにおける例外の種類レベルでの枝刈り可能性を $\theta^{(R)}(v_c, p) \in \{0,1\}$ とする．また，パターンエンティティレベルでの枝刈り可能性を $\theta(v_c, p) \in \{0,1\}$ とする．パターン p において，パターンレベルでの枝刈り可能性を $\theta(p) \in \{0,1\}$ とする．これらの枝刈り可能性の詳細な求め方は 5.2.3 節にて後述する．

5.2.3 枝刈り器のアルゴリズム

枝刈り器 $\mathrm{PR}(p, G, S_{min})$ は，パターン p，知識グラフ G を入力として，$\theta(p)$ および各 $v_c \in \mathcal{V}_p$ における $\theta(v_c, p), \theta^{(L)}(v_c, p), \theta^{(U)}(v_c, p), \theta^{(R)}(v_c, p)$ を出力する．まず，p 中のすべてのパターンエンティティ $v_c \in \mathcal{V}_p$ についてループする．ループ内では，エンティティラベルにおける上界スコア $U^{(L)}((v_c, p), G)$ を計算し， top-k の例外スコア S_{min} 未満であれば，$\theta^{(L)}(v_c, p) = 1$ とすることにより，エンティティラベルにおける例外の種類レベルでの枝刈りを行う．同様に，$U^{(L)}((v_c, p), G)$ の計算により固有エンティティにおける例外の種類レベルでの枝刈り，$U^{(R)}((v_c, p), G)$ の計算によりリレーションにおける例外の種類レベルでの枝刈りを行い，$\theta^{(U)}(v_c, p), \theta^{(R)}(v_c, p)$ の値を求める．次に

Algorithm 2: パターン発見器 $\mathrm{PD}(n, G, P[n])$

```
   input  : The number of discovered pattern triples n
1  if n = 0 then
2      for f ∈ L_V ∪ V do
3          V_p ← {f}, E_p ← φ
4          PS[0] ← PS[0] ∪ {(V_p, E_p)}
5  else
6      for p ∈ PS[n − 1] do
7          for v_c ∈ V_p do
8              for (r^(D), v_π, π) ∈ Π(v_c, p) do
9                  if θ(p, G) · θ(π, G) = 0 then
10                     V_p' ← V_p ∪ {v_π}
11                     E_p' ← E_p ∪ {(v_c, r^(D), v_π)}
12                     PS[n] ← PS[n] ∪ {(V_p', E_p')}
```

Algorithm 3: 例外検出

```
   input  : Knowledge graph G, The number of triples or
            entities to output k, The maximum number of
            pattern triples in a pattern N
   output : The content of OSR
1  for n = 0 to N do
2      PS[n] ← φ
3  PD(0, G, P[0])
4  for m ∈ [0, N] do
5      for p ∈ PS[m] do
6          (θ(p), (θ(v_c, p), θ^(L)(v_c, p),
            θ^(U)(v_c, p), θ^(R)(v_c, p)) for v_c in V_p)
            ← PR(p, G, OSR[k].Score)
7          for v_c ∈ V_p do
8              if θ(v_c, p) = 0 then
9                  OSC ((v_c, p), G, OSR[k].Score)
10     if m ≠ N then
11         for p ∈ PS[0] ∪ PS[m] do
12             (θ(p), (θ(v_c, p), θ^(L)(v_c, p), θ^(U)(v_c, p),
                θ^(R)(v_c, p)) for v_c in V_p)
                = PR(p, G, OSR[k].Score)
13         PD(m + 1, G, PD[m + 1])
14 return OSR
```

$\theta(v_c, p) = \theta^{(L)}(v_c, p) \cdot \theta^{(U)}(v_c, p) \cdot \theta^{(R)}(v_c, p)$ の計算により，v_c に対してパターンエンティティレベルでの枝刈りを行う．最後に，$\theta(p) = \prod_{v_c \in \mathcal{V}_p} \theta(v_c, p)$ の計算により，p に対してパターンレベルでの枝刈りを行う．

5.3 パターン発見器

パターン発見器では，知識グラフ上に存在するパターンを発見する．パターントリプル数 0 のパターンは，全探索による素朴な手法で発見する．パターントリプル数 1 以上のパターンは，枝刈りを用いた効率的な手法により，発見済みのパターンにパターントリプルを追加することで発見する．

Algorithm 2 にパターン発見器のアルゴリズムを示す．なお，アルゴリズム中の PS[i] は，パターンストアで格納しているパターントリプル数 i のパターンの集合である．$n = 0$ の場合，G に存在するパターントリプル数 0 のパターンを全て発見し，それらをパターンストアに格納する（2–4 行目）．

$n \neq 0$ の場合，次のようにしてパターン発見を行う（6–12 行目）．まず，パターンストア中のパターントリプル数 $n-1$ のパターン p を走査する．そして，p のパターンエンティティ $v_c \in \mathcal{V}_p$ を走査する．そして，$\Pi(v_c, p)$ の要素を走査する．ここで，$\Pi(v_c, p)$ は，あるエンティティ $v \in \mathcal{M}((v_c, p), G)$ において，有向リレーションタイプ $r^{(D)}$ が v と接続し，かつ v と $r^{(D)}$ によって接続するあるエンティティ v' が $v' \in \mathcal{M}((v_c', p'), G)(\mathcal{T}_{p'} = \phi)$ となるような全ての組 $(r^{(D)}, v_c', p')$ の集合である．$\Pi(v_c, p)$ を走査するループ内では，p と π のいずれかの枝刈りが不可能なときのみ，π を構成するパターンエンティティ v_π と，パターントリプル $(v_c, r^{(D)}, v_\pi)$ を p に追加した新たなパターン $p^{(new)}$ を発見する．

5.4 例外スコア計算器

例外スコア計算器 $\mathrm{OSC}(C, G, S_{min})$ は，コンテキスト C，知識グラフ G を入力とする．まず，C にマッチする全てのエンティティ v と，v を含む全てのトリプルに対して $S(v, C, G), S(t, C, G)$ を計算する．この計算の際，枝刈り結果を用いて以下のように効率的に計算する．(1) 式または (2) 式の例外スコアを計算する際に，$\theta^{(L)}(v_c, p) = 1$ の場合は第 1 項を，$\theta^{(U)}(v_c, p) = 1$ の場合は第 2 項を，$\theta^{(R)}(v_c, p) = 1$ の場合は第 3 項を最小値の 0 として計算することで，その項の計算を省略する．そして，そのスコア S が現在の例外スコアランキング top-k のスコア S_{min} 以上である場合，S，v または t，C と，その例外スコアの計算をする際に用いた出現分布を，例外スコアランキングに格納する．

5.5 アルゴリズム

まずパターントリプル数 0 のパターンを発見する．そして，適宜枝刈りを行いながら，例外スコア計算とパターン発見を行う．パターン発見の際は，既に発見されたパターンにパターントリプルを追加することによりパターンを新たに発見する．これらを，対象とするパターンのパターントリプル数を 1 ずつ増やしながら行う．

Algorithm 3 に例外検出のアルゴリズムを示す．なお，アルゴリズム中の OSR[k].Score は，top-k の例外スコアである．まず，パターントリプル数 0 のパターンを全て発見し，それらをパターン発見器に格納する（3 行目）．

次に，m を 0 から N までループする．ループ内では，まず，パターントリプル数 m のパターンから構成されるコンテキストにマッチするエンティティやそれを含むトリプルの例外スコアの計算を行う（5–9 行目）．このとき，走査中のパターンに対して枝刈りを行った後，そのパターンの各パターンエンティティに対してパターンエンティティレベルでの枝刈りが不可能なときのみ例外スコアの計算を行う．

次に，パターントリプル数 0 または m のパターンに対し枝刈りを行う（11–12 行目）．これは，枝刈り結果をパターン発見の際に使用するためである．枝刈りの後，パターントリプル数 $m+1$ のパターンを発見する（13 行目）．最後に，探索すべきパターンを全て終えたときの例外スコアランキングの内容を出力する (14 行目).

6 実験

本章では，評価実験について説明する．6.1 節で実データから作成したデータセットについて説明する．6.2 節で提案手法の検出効率の検証結果を示す．6.3 節で発見した例外的事実の有用性の検証結果を示す．

実験は， AMD EPYC 7542 32-Core Processor (コア数 64C, クロック数 2.9GHz, CPU 数 2P)，2048GB メモリを搭載した Ubuntu 18.04 LTS Server 上で実行した．全ての

表 1: 各知識グラフのサイズ

要素	NELL	DBpedia
エンティティ数 $\lvert\mathcal{V}\rvert$	46682	1477796
トリプル数 $\lvert\mathcal{E}\rvert$	231634	2920168
エンティティラベル数 $\lvert\mathcal{L}_V\rvert$	266	239
リレーションタイプ数 $\lvert\mathcal{L}_E\rvert$	821	504

表 2: NELL の各種統計

要素	最大値	最小値	平均値
次数	3614	0	9.92
エンティティあたりのラベル数	7	1	1.53
ラベル毎のエンティティ数	3640	1	268.39

表 3: DBpedia の各種統計

要素	最大値	最小値	平均値
次数	67976	0	3.95
エンティティあたりのラベル数	5	1	2.72
ラベル毎のエンティティ数	413404	1	16810.8

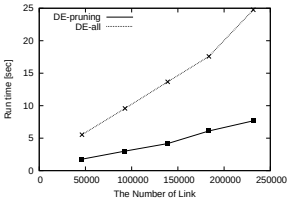


図 5: NELL のトリプル数の実行時間への影響（$N = 0$）

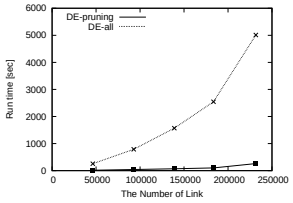


図 6: NELL のトリプル数の実行時間への影響（$N = 1$）

アルゴリズムは C++ を利用して実装・実行した．

6.1 データセット

データセットとして，知識グラフ NELL [5] および DBpedia [6] を使用した．[1]NELL は Web 上のデータの反復的な学習によって知識ベースを生成する技術である．このデータセットは，1115 回の反復により生成されている [12]．DBpedia は Wikipedia のデータを RDF トリプルに変換したものである．表 1 にこれらの知識グラフのサイズを，表 2 および表 3 にそれぞれ NELL と DBpedia に関する各種統計を示す．

6.2 検出効率

本節では，実行時間の計測によって提案手法の検出効率を検証する．

実験設定　検出効率に関する実験として，トリプル数を変化させて実行時間を計測する実験および k を変化させて実行時間を計測する実験を行った．トリプル数を変化させる実験では，データセットのトリプル数の割合を 20%, 40%, 60%, 80%, 100%と変化させながら実行した．パラメータは $k = 100$ とし，N は 0 と 1 の二通りを用いる．k を変化させる実験では，データセットとして NELL のみを用いて，k を $10, 50, 100, 150, 200, 250, 300$ と変化させながら実行した．N は 1 とした．

トリプル数による実行時間の変化　図 5 および図 6 に N を 0 および 1 とした場合の NELL における 2 回の平均実行時間をそれぞれ示す．図 7 および図 8 に N を 0 および 1 とした場合の DBpedia における 2 回の平均実行時間をそれぞれ示す． Algorithm 3 において枝刈りに関する処理を除いた全体のアルゴリズムを DE-all，提案手法を DE-pruning とし，両者を比較する．なお，実行は 24 時間で打ち切ったため，抜けているプロットが存在する．

1） いずれも https://github.com/GemsLab/KGist にて公開されている．

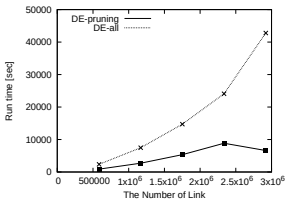


図 7: DBpedia のトリプル数の実行時間への影響（$N = 0$）

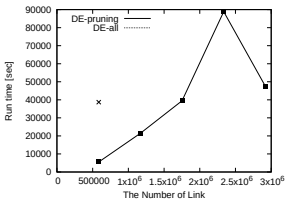


図 8: DBpedia のトリプル数の実行時間への影響（$N = 1$）

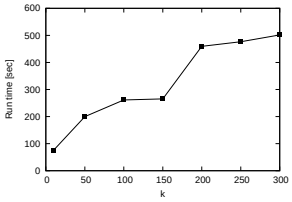


図 9: k の実行時間への影響

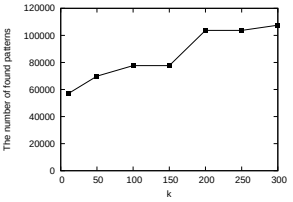
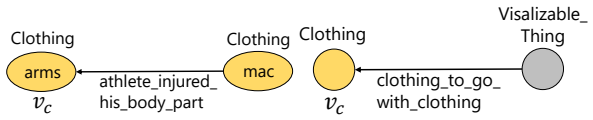


図 10: k の発見されるパターン数への影響



(a) 例外トリプル　(b) コンテキスト

(c) $\mathcal{M}(C, G)$ の各エンティティとラベルが Clothing であるエンティティとの間の有向リレーションタイプの出現分布

有向リレーションタイプ	数
clothing_to_go_with_clothing	6499
clothing_to_go_with_clothing^{-1}	6499
inverse_of_athlete_injured_his_body_part	1
athlete_injured_his_body_part^{-1}	1

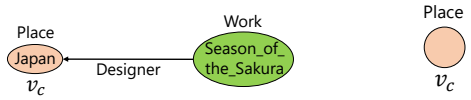
図 11: NELL の実験による出力結果

k による実行時間の変化　図 9 に 5 回の平均実行時間を，図 10 に発見されたパターン数を示す．図 9，図 10 から，k が小さい方が発見されるパターン数が少なく，枝刈りが効果的であることがわかる．

6.3 例外的事実の有用性

本節では，発見した例外的事実の内容の分析によって，例外的事実の有用性を検証する．

データセットは NELL とし，パラメータは $k = 100, N = 1$ とした．図 11 に出力された例外的事実の一例を示す．図 11a のトリプル，図 11b のコンテキスト，表 11c の



(a) 例外トリプル　(b) コンテキスト

(c) $\mathcal{M}(C,G)$ の各エンティティと有向リレーションタイプ Designer^{-1} で結合しているエンティティの出現分布

エンティティ	数
エンティティなし	413403
Seanson_of_the_Sakura	1
Yo-Jin-Bo	1

図 12: DBpedia の実験による出力結果

出現分布の意味は，服の arms が服の mac によって傷つけられた体の部位であるという内容のトリプルは例外であり，その理由は，視認できるものに合う服と服のエンティティの間に傷つけられた体の部位という有向リレーションタイプがあまり存在しないためであることを表す．使用した知識グラフのデータには，ラベルが body_part である arms のエンティティと，ラベルが athlete である mac のエンティティが別に存在した．そのため，体の部位である arms を服ブランド arms と誤り，人名である mac を mac と誤ったことにより，リレーションを張り間違えたと考えられる．上述の出力の他にも，服の bottom を焼き菓子の商品である bottom と誤ったためと考えられるような出力内容など，誤り検出に適用可能な例外的事実が多く検出された．

次に，データセットとして DBpedia を用いた実験の結果を示す．パラメータは同じである．図 12 に出力された例外的事実の一例を示す．図 12a のトリプル，図 12b のコンテキスト，表 12c の出現分布の意味は，場所の Japan は作品の Season_of_the_Sakura の作者であるという内容のトリプルは例外であり，その理由は，場所のエンティティは作者という有向リレーションタイプとあまり結合していないためであることを表す．Designer^{-1} の有向リレーションタイプと接続するエンティティのラベルとして Person 等が存在したが，Place はこの 1 つしか存在しなかった．これは，人としての作者の情報が足りず，作者として国を当てはめざるを得なかったためと考えられる．

上述の出力の他にも，Rome が Daniel_Comboni を列福させたが，誰かを列福させる国はあまり存在しないといった例外的事実が検出された．しかし，図 12b のコンテキスト以外を出力とする例外的事実は出力されなかった．これは，図 12b のコンテキストにマッチするエンティティの数が他のエンティティより多く，(6) 式の a_i や a_ϕ の値が大きくなることによって (6) 式の値も大きくなったためと考えられる．

7 結論

本研究では，知識グラフにおける例外検出手法を提案した．提案手法では，様々な隣接要素の出現分布を基に例外スコアを計算し，上界スコアを用いた効率的な探索を可能にした．実験では実データを用いて検出効率と例外的事実の有用性の評価を行った．

今後の展望としては，例外スコアの評価性能の向上および実行時間を削減し，より実用的な知識グラフの例外検出技術の開発を目指す．現在，全てのエンティティを対象にして例外検出をおこなっている．しかし，これではユーザの無関心なジャンルの例外的事実も検出されてしまい，実行時間の長さに対して利益が少なく，また top-k が全て無関心なジャンルで埋め尽くされる可能性もある．そこで，例外対象や探索するパターンに対して条件付けを行うことで，ユーザの興味のある例外的事実をより短い時間で検出する例外検出技術の開発を行う．DBpedia における実験では，$\mathcal{M}(C,G)$ が多いコンテキスト C にマッチするエンティティばかりが例外として検出されていた．そこで，例外スコアの上限を一定にする定数をコンテキストごとに設け，それを例外スコアにかけることで，様々なコンテキストにマッチするエンティティを例外として検出することを目指す．また，並列化や近似解によるさらなる実行時間の削減も目指す．

謝辞

本研究は JST さきがけ JPMJPR18UD および JSPS 科研費 JP20H00583 の支援によって行われた．ここに記して謝意を表す．

参考文献

[1] Matteo Lissandrini, Davide Mottin, Themis Palpanas, and Yannis Velegrakis. Graph-query suggestions for knowledge graph exploration. In *Proceedings of the WWW*, pp. 2549–2555, 2020.

[2] Amit Singhal. Introducing the knowledge graph: things, not strings. https://blog.google/products/search/introducing-knowledge-graph-things-not/, (Retrieved on June 24, 2022).

[3] Gensheng Zhang, Damian Jimenez, and Chengkai Li. Maverick: Discovering exceptional facts from knowledge graphs. In *Proceedings of the SIGMOD*, pp. 1317–1332, 2018.

[4] Yueji Yang, Yuchen Li, Panagiotis Karras, and Anthony KH Tung. Context-aware outstanding fact mining from knowledge graphs. In *Proceedings of the KDD*, pp. 2006–2016, 2021.

[5] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the AAAI*, pp. 4444–4451, 2010.

[6] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pp. 722–735. Springer, 2007.

[7] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. Focused clustering and outlier detection in large attributed graphs. In *Proceedings of the KDD*, pp. 1346–1355, 2014.

[8] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball:spotting anomalies in weighted graphs. In *Proceedings of the PAKDD*, pp. 410–421, 2010.

[9] U Kang, Jay-Yoon Lee, Danai Koutra, , and Christos Faloutsos. Net-ray: Visualizing and mining billion-scale graphs. In *Proceedings of the PAKDD*, pp. 348–361, 2014.

[10] Jing Gao, Feng Liang, Wei Fan, Chi Wang, Yizhou Sun, and Jiawei Han. On community outliers and their efficient detection in information network. In *Proceedings of the KDD*, pp. 813–822, 2010.

[11] Manish Gupta, Arun Mallya, Subhro Roy, Jason H. D. Cho, and Jiawei Han. Local learning for mining outlier subgraphs from network datasets. In *Proceedings of the SDM*, pp. 73–81, 2014.

[12] Caleb Belth, Xinyi Zheng, Jilles Vreeken, and Danai Koutra. What is normal, what is strange, and what is missing in a knowledge graph: Unified characterization via inductive summarization. In *Proceedings of the WWW*, pp. 1115–1126, 2020.