ESPRESSO: Explaining Relationships between Entity Sets

ABSTRACT

Analyzing and explaining relationships between entities in a knowledge graph is a fundamental problem with many applications. Prior work has been limited to extracting the most informative subgraph connecting two entities of interest. This paper extends and generalizes the state of the art by considering the relationships between two sets of entities given at query time. Our method, coined ESPRESSO, explains the connection between these sets in terms of a small number of relatedness cores: dense sub-graphs that have strong relations with both query sets. The intuition for this model is that the cores correspond to key events in which entities from both sets play a major role. For example, to explain the relationships between US politicians and European politicians, our method identifies events like the PRISM scandal and the Syrian Civil War as relatedness cores. Computing cores of bounded size is NP-hard. This paper presents efficient approximation algorithms. Our experiments with real-life knowledge graphs demonstrate the practical viability of our approach and, through user studies, the superior output quality compared to state-of-the-art baselines.

1. INTRODUCTION

With the growth of large-scale knowledge bases (modeled as graphs), there is a need for algorithmic solutions that facilitate the semantically meaningful analysis of relationships between entities in the knowledge graph [15, 16, 30, 38]. We focus on such an analysis over knowledge graphs, wherein (semantically typed) entities are connected via semantic relationships – e.g., Google Knowledge Graph, the Facebook Entity Graph, and the Web of Linked Open Data which is centered around public knowledge bases such as DBPedia [6], Freebase [9], and Yago [24, 37].

In this paper, we introduce and give a scalable solution to the problem of explaining the relationship between two sets of entities in a knowledge graph. Given a knowledge graph $K$, consisting of vertices corresponding to entities that are connected via semantically typed edges, together with two subsets of entities $Q_1, Q_2$, we are interested in characterizing the relationship between the entity sets $Q_1$ and $Q_2$.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CIKM ’16, October 24–November 28, 2016, Indianapolis, IN, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4073-1/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2983323.2983778

As an example, the goal of a relationship characterization algorithm over a knowledge graph containing facts about real-world entities – such as people, countries, organizations, etc. – is to answer questions of the form «Which European politicians are related to politicians in the United States, and how?» or «How can one summarize the relationship between China and countries from the Middle East over the last few years?»

We develop a novel algorithm – coined Espresso – to compute semantically meaningful substructures (so-called relatedness cores) from the knowledge graph as answers to above questions. In our setting, a question is specified by means of two sets of query entities. These sets of entities (e.g. European politicians or United States politicians) can be determined by an initial query, e.g., expressed in the SPARQL language for RDF data, or simply enumerated. As a first step, we analyze the (indirect) relationships that connect entities from both sets (e.g., membership in organizations, statements made on TV, etc.), generate an informative and concise result, and finally provide a user-friendly explanation of the answer. As output we aim to return concise subgraphs that connect entities from the two sets and explain their relationships.

As an example, consider the graph depicted in Figure 1, which demonstrates the desired output of a relationship characterization system. Here, we are interested in characterizing the political relationship between European countries and the United States. For this purpose, compact and coherent subgraphs from the proximity of important politicians from the specified countries are displayed. Each such subgraph corresponds to a key event that is highly relevant to at least one entity from each of the two input sets. These subgraphs form the core of the relationship explanation; we thus call them relatedness cores. The full answer to the user query can be derived by connecting relatedness cores with the entities in the two input sets.

1.1 Problem Characteristics

Previous work on relationship explanation over graphs has mainly focused on computing the “best” subgraph providing a connection...
among the entities in a single query set \[15, 16, 30, 38\]. These algorithms operate typically in two stages: first, vertices and/or edges are assigned a score signifying their informativeness or relatedness to the query entities, and in the second stage, a solution subgraph connecting the query vertices is constructed based on the assigned scores. Although we continue to view the problem of explaining the relationship between entities as an issue of computing the informative subgraph, in this paper we deviate from previous approaches, since we are interested in the relationship between two sets of query entities, \(Q_1, Q_2\), rather than among a set of entities. Our output is not focused on a single coherent subgraph, but aims to find multiple substructures in the knowledge graph that are highly informative.

To this end, we adapt and extend the CePS (Centerpiece Subgraphs [38]) framework for computing good subgraphs connecting entities contained in a single query set to the case of two query sets in knowledge graphs. As a building block along these lines, we define a notion of relationship centers, intermediate vertices that play an important role in the relationship between entities from either set. Relationship centers play a role similar to centerpiece vertices in CePS, with the difference that relationship centers are informative vertices (by entity type restriction) for the relationship between two sets rather than among a single set. To answer a query, we identify a set of relationship centers, connect them to (subsets) of the query entities, and subsequently expand the center vertices into relatedness cores: concise subgraphs that represent key events connected to both input sets of entities. This outlined method can be seen as the approximation of a combinatorial optimization problem specified over the underlying graph and input entity sets (to be explained in Section 3), extended to incorporate considerations regarding the semantics of the generated solution, which cannot be easily compressed merely in terms of graph-theoretic properties.

1.2 Contribution

The salient contributions we make in this paper are:

1. We introduce a model for the novel problem of explaining relationships between two intensionally specified sets of entities, in contrast to the previously studied problem of explanations for a pair of single entities or among a single set of query entities,

2. adapt and extend the CePS framework into a heuristic algorithm for computing explanations of the relationship between two sets (based on the notions of relationship centers and relatedness cores) in a scalable manner,

3. discuss additional scenarios such as the integration of user-specified constraints on the computed solutions and the integration of temporal information, and,

4. present an experimental evaluation based on user assessments on the usefulness of the generated explanations, as well as experiments highlighting the efficiency of the heuristic algorithm.

We compute the solutions over an enriched entity-relationship graph derived from existing knowledge bases and integrating additional data sources. The remainder of the paper is organized as follows. In Section 2 we discuss the knowledge graph we use to compute relationship explanations. Section 3 presents our computational model. In Sections 4 and 5 we present our heuristic algorithm for scalable computation of relationship explanations, introducing the two major building blocks of our solution, relationship centers and relatedness cores. Section 6 extends our setting to consider temporal aspects in the computation of entity relatedness. Section 7 discusses related work. Section 8 presents experimental results, followed by a summary of the paper.

2. ESPRESSO KNOWLEDGE GRAPH

The knowledge graph we use as input to our relationship characterization algorithm – in this paper referred to as Espresso Knowledge Graph – is derived from the YAGO2 [24] and Freebase [9] knowledge bases. More specifically, the set of entities we model as vertices are exactly the entities present in the intersection of YAGO2 and Freebase. This way, we can discard many concepts present in YAGO2 that do not correspond to actual entities, for example overview pages from Wikipedia such as discographies, summaries (2013 in film), etc. Two entities \((e_1, e_2)\) in the Espresso Knowledge Graph are connected via an undirected edge, if the corresponding Wikipedia article pages describing the respective entities contain an intra-wiki link in either direction. Every edge is assigned the relationship label relatedTo as well as additional labels, corresponding to the relation names for each fact contained in YAGO2 between the respective entities. The Espresso Knowledge Graph contains a total of 3,674,915 vertices (corresponding to entities), connected via 57,703,180 relationships labeled with one of 30 possible relationship names.

We enrich this knowledge graph by integrating several additional data sources, including (i) edge weights signifying the relatedness between entities, derived from structural properties (unlink overlap [33]), textual descriptions of the entities based on the Wikipedia article text [23], and co-occurrence in the ClueWeb12 corpus [19], (ii) semantic types associated with the entities, derived from YAGO2 (Wikipedia categories, WordNet classes) and Freebase (types). Every entity is assigned one or more of the 508,356 YAGO types and one or more of the 7,513 Freebase types. On average, an entity is associated with 15.8 YAGO types and 3.3 Freebase types, translating to a total of 58,023,593 entity-YAGO type and 12,278,102 entity-Freebase type relationships, (iii) the popularity of individual entities over time, extracted from the page view statistics of the respective Wikipedia pages\(^1\), a total of 2,827,668,135 triples of the form (entity,day,views) reflecting the page view counts of each entity in daily granularity in the time frame 01/01/2012 to 07/31/2014. The Espresso algorithm described later in this paper relies on identifying query-relevant entities of informative types, e.g. – as used in the experimental evaluation of this paper – entities of type event. In principle, events can be identified by considering all entities typed Event (WordNet class) in YAGO or /time/event in Freebase. However, in order to increase both precision as well as recall, we employ a machine learning approach to identify entities of the event type. For this purpose, we have trained a linear SVM classifier by manually identifying 1,786 entities corresponding to real-world events from a pool of 28,000 training examples. The features used to classify an entity are the entity name and short snippets describing the entities (extracted from Freebase). We use the model to classify each of the 3.6 million entities as either event or not event, resulting in a total of 89,321 entities marked as event\(^2\).

3. COMPUTATIONAL MODEL

We define the notion of a knowledge graph as described above formally as follows: A knowledge graph, denoted by \(K = (V, E, \ell_V, \ell_E, T, R)\), is a labeled graph with a set \(V\) of entities as vertices, a set \(E\) of relationships as edges, and two labeling functions \(\ell_V : V \rightarrow 2^T\) and \(\ell_E : E \rightarrow 2^R\) that assign to each vertex a set of type names (from a set \(T\) of possible types) and to each edge a set of relation names (from a set \(R\) of possible relations) as labels. Each edge \(e\) and each vertex \(v\) can optionally have weights \(\omega(e)\) and \(\omega(v)\). As an example, the vertex Hillary Clinton in Figure 1 may have type labels female politician, US Democratic Party member, US Secretary

\[\text{http://dumps.wikimedia.org/other/pagecounts-ez/}\]

\[\text{http://espresso.mpi-inf.mpg.de/}\]
of State, First Lady of the United States, and more, and the edge between Hillary Clinton and Barack Obama could have relation labels competed with (in US presidential primaries) and served under (in the US government).

Starting with two sets \( Q_1 \) and \( Q_2 \) of entities, such as United States politicians and European politicians, the explanation of their relationships aims to identify one or more key events that connect the two sets. Key events are themselves groups of strongly interrelated entities. We call the corresponding subgraphs \( S_i \) relatedness cores, or cores for short. As a general framework, we place the following desiderata to exemplify what constitutes a “good” core:

**Informativeness.**

1. Critical: The core \( S_i \) is important, i.e. it describes an important event or topic.
2. Comprehensive: The core \( S_i \) is self-explanatory, i.e. gives insight into the topic without requiring further explanation.
3. Compact: The core \( S_i \) does not overload the user with information by satisfying an upper bound on the size, i.e. the number of contained vertices.
4. Coherent: The core \( S_i \) describes a single event or topic rather than a mixture of topics. For this purpose, the entities contained in the core should be strongly interrelated, corresponding to a subgraph with high pair-wise edge weights or high degrees within \( S_i \).

**Relevance.**

1. The core is relevant to some or all query entities, i.e. \( S_i \) should be highly related to both entity sets \( Q_1 \) and \( Q_2 \), for example in terms of edges or paths between \( S_i \) and each of the two query sets, total or average edge weight of paths connecting them, or another way that captures relatedness between vertices \( v \in S_i \) and query vertices \( q \in Q_1 \cup Q_2 \).

2. Temporal relevance: The topic was important/the event happened during a user-specified time interval.

One way of formalizing these desiderata into a computational model is to aim for identifying one or more subgraphs as relatedness cores with high edge weights (i) within each subgraph and (ii) regarding the connections to each of the two input sets. When considering paths between some core \( S_i \) and a query set \( Q_1 \) or \( Q_2 \), we need to combine edge weights by an aggregation function like average, maximum, or minimum applied to the shortest path (or shortest paths) between vertices in \( S_i \) and all or some of the vertices in \( Q_1 \) or \( Q_2 \). We denote the path score between vertices \( x \) and \( y \) as \( \omega (x \leftrightarrow y) \). Formally, we cast these ideas into the following *edge-weighted relatedness cores problem.*

**Definition 1.** Given an edge-weighted knowledge graph \( K = (V, E, \ell_V, \ell_E, T, R) \), two query sets \( Q_1 \subset V \) and \( Q_2 \subset V \) and a budget \( B \), we define the Edge-Weighted Cores Problem as the problem of computing up to \( k \) connected subgraphs \( S_1 = (V_1, E_1), \ldots, S_k = (V_k, E_k) \) such that

\[
\sum_{i=1}^{k} \sum_{e \in E_i} \omega(e) + \sum_{x \leftrightarrow y \in V_i} \sum_{x \in Q_1, y \in V_i} \omega(x \leftrightarrow y) \quad \sum_{x \leftrightarrow y \in V_i} \sum_{x \in Q_2, y \in V_i} \omega(x \leftrightarrow y)
\]

is maximized with \( |V_i| \leq B \) and each \( V_i \cap Q_1 \neq \emptyset \neq V_i \cap Q_2 \). □

Intuitively, in this setting we aim to find up to \( k \) coherent cores that explain the relationship between \( Q_1 \) and \( Q_2 \), while observing a size budget for each core. The density of each \( S_i \) is measured by the total weight of edges in \( S_i \), and the relatedness to \( Q_1 \) and \( Q_2 \) is measured by the total score of paths that connect vertices in \( S_i \) with vertices in \( Q_1 \) and \( Q_2 \), respectively. This problem formulation is well-suited for the scenario where we seek an explanation involving all entities from both query sets. We can easily show the NP-hardness of this problem by constructing an equivalent problem – computing the densest subgraph with fixed cardinality – a known NP-hard [17] problem. Due to lack of space, we omit the details of this construction.

Above problem formulation considers paths (or, more precisely, distances) between the query and core vertices to assess the relevance. In many cases, it makes more sense to assess relevance by more involved measures (such as proximity derived from random walks). We now consider the case where the individual core vertices are assigned a score signifying their relevance. This leads to the following, combined problem with a tunable coefficient \( \beta \) to balance between relevance and coherence.

**Definition 2.** Given an edge-weighted knowledge graph \( K = (V, E, \ell_V, \ell_E, T, R) \), two query sets \( Q_1 \subset V \) and \( Q_2 \subset V \) and a budget \( B \), the Edge- and Vertex-weighted Cores Problem is to compute up to \( k \) connected subgraphs \( S_1 = (V_1, E_1), \ldots, S_k = (V_k, E_k) \) such that

\[
\sum_{i=1}^{k} \beta \sum_{e \in E_i} \omega(e) + (1 - \beta) \sum_{v \in V_i} \omega(v)
\]

is maximized with \( |V_i| \leq B \) and each \( V_i \cap Q_1 \neq \emptyset \neq V_i \cap Q_2 \). □

By encoding the relevance of core vertices to the query entities in the form of vertex weights, we can support different scenarios, including the case where only subsets of the query sets have to be related to a core.

The special case with \( \beta = 0 \) (i.e., vertex weights only) is also known to be NP-hard, following from the intuition given for the hardness of the edge-weighted core problem and the fact that the general problem of computing the heaviest vertex-weighted tree spanning \( k \) vertices has been shown to be NP-hard [18].

### 4. RELATIONSHIP CENTERS

The idea of the Espresso algorithm is to identify relatedness cores using a two-stage algorithm. In the first stage, we identify a set of relationship centers, intermediate vertices that play an important role in the relationship – similar to centerpiece vertices in CePS [38]. These centers are subsequently connected to the query vertices. In the second stage, the neighborhoods of the best relationship centers are expanded, in order to obtain the relatedness cores, concise subgraphs that represent key events connected to both input sets of entities. In this section, we discuss how the relationship centers – i.e. vertices that exhibit a high potential for explaining the relationship between the two input sets of entities – can be computed.

Intuitively, the goal is to identify vertices in the knowledge graph that have strong relations with vertices in both input sets. We adopt the idea of identifying central vertices by performing random walks over the graph [38]. For this purpose, we operate on an entity-relationship graph that we derive from the given knowledge graph and additional statistics. The importance of a vertex for explaining the relationship between the query sets is quantified by assigning each vertex in the graph a numeric score based on the random walk processes. For the example of explaining the connections between European and US politicians, the PRISM surveillance program, G8 summits, etc. represent entities for which we would desire a high vertex score. In the
following, we first define the relatedness graph, and then present our method for random-walk based scoring of vertices.

**Entity-Entity Relatedness Graph.** Our relationship center extraction algorithm operates on the edge-weighted knowledge graph described in Section 2. For edge weights (sim), we considered the popular inlink overlap measure [33] (abbreviated as MW) as well as recently proposed partial keyphrase overlap measure [23] (abbreviated as KORE).

Given the relatedness measure, we obtain the weighted adjacency matrix of the graph, given by

\[ M \in \mathbb{R}^{n \times n}, \quad M[i, j] = \begin{cases} \text{sim}(v_i, v_j) & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise}. \end{cases} \]  

Equation (1)

Following the intuition given by [38], we construct the transition matrix for the random walks by introducing \( \hat{M} = D^{-1/2}MD^{-1/2} \), where \( D \) is the diagonal matrix with entries \( D[i, i] = \sum_{k=1}^{n} M[i, k] \). Note that with this use of the normalized graph Laplacian for constructing \( \hat{M} \), the column vectors are no longer stochastic [39], so we need to add a normalization step after executing a random walk.

**RWR for Relationship Centers.** In the next subsections we discuss how the transition matrix is used to conduct random walks with restart (RWR) in order to identify vertices relevant to the query. We first describe how CePS employs RWRs to identify centerpiece vertices that are related to query vertices belonging to a single set \( Q \).

**The CePS approach.** The CePS algorithm executes one random walk with restart (RWR) from each query vertex \( q \in Q \). As a result, each vertex \( v \in V \) in the graph is assigned \( |Q| \) scores, \( r_q(v) \). The random walk computation from \( q \) in \( Q \) works as follows: We set the starting probability vector \( s \in \mathbb{R}^n \) to \( s[q] = 1 \) and \( s[v] = 0, v \in V \setminus \{q\} \). At vertex \( v \), with probability \( \alpha \hat{M}[v, w] \), we walk to a neighboring vertex \( w \). \( \hat{M}[x, y] \) denotes the transition probability from vertex \( x \) to vertex \( y \). With probability \( 1 - \alpha \), we jump back to start vertex \( q \). To obtain the steady-state probabilities, we iterate the equation

\[ x_{i+1} = (1 - \alpha)x + \alpha \hat{M}x, \quad x_0 = s. \]  

Equation (2)

until convergence or for a predetermined number of iterations. The score \( r_q(v) \) then corresponds to the steady-state probability \( x[v] \). The individual scores are aggregated over an overall score of the vertex \( v \), depending on the desired scenario. [38] distinguish between the scenarios AND (all query vertices should be highly related \( v \)), OR (at least one query vertex should be highly related to \( v \)) and Soft-AND (at least \( k \) of the \( |Q| \) vertices should be highly related to \( v \)). Since the score \( r_q(v) \) is interpreted as the probability for a random walk particle starting at \( q \) to be located at \( v \), in the AND-scenario the scores (probabilities) are simply multiplied to aggregate a total score:

\[ \sigma_{\text{CePS-AND}}(v) = \prod_{q \in Q} r_q(v). \]  

Equation (3)

Conversely, for the OR-scenario, the aggregated score corresponds to the probability that one or more random walk particles from the individual random walks are located at \( v \):

\[ \sigma_{\text{CePS-OR}}(v) = 1 - \prod_{q \in Q} (1 - r_q(v)). \]  

Equation (4)

With this approach, CePS can identify vertices with high proximity to all or at least one (at least \( k \)) of the query vertices. To compute the corresponding scores, it requires to perform \(|Q|\) random walks with restart over the input graph.

We now discuss how this framework can be extended in order to identify vertices relevant to two sets of query vertices.

**Relationship Centrality.** We present an extension of the CePS method to find vertices highly related to the two sets \( Q_1, Q_2 \subseteq V \). Suppose all query entities from either set should be related to the center. Then, an appropriate score of a vertex with respect to the query sets is given by

\[ \sigma_{\text{CePS-AND}}(v) = \prod_{q \in Q_1} r_q(v) \prod_{q \in Q_2} r_q(v) = \prod_{q \in Q} r_q(v). \]  

Equation (5)

i.e., in this case we can directly apply the previous score aggregation for the case of a single query set. This scenario requires however, that all query entities should be related to the central vertices. In many practical settings, including our envisioned applications, it is a far more common assumption that only certain subsets of the query entities are related. To this end, we propose the following generalization of the CePS scoring mechanism to deal with two sets of query entities. Continuing the description above, in this scenario we quantify the relatedness of a vertex to the query sets \( Q_1, Q_2 \) as the probability that at least one random walk particle from either set is located at vertex \( v \). This is expressed in the formula

\[ \sigma_{\text{CePS-OR}}(v) = \left[ 1 - \prod_{q \in Q_1} (1 - r_q(v)) \right] \left[ 1 - \prod_{q \in Q_2} (1 - r_q(v)) \right]. \]  

Equation (6)

We can expect this approach to capture vertices that are in close proximity to subsets of the two query sets. However, while Equation 6 is an appropriate generalization of the original CePS approach to sets, we are still facing the problem of having to conduct \(|Q|\) random walks. To this end, [38] and [39] propose substantial improvements over straightforward implementations of RWRs (like partitioning and low-rank matrix approximation), however, very large graphs in combination with many query vertices can remain challenging, especially in our envisioned scenario of an interactive, user-facing system.

To overcome this problem, we propose a faster approach to identify candidates for relationship centers, i.e., vertices in close proximity to subsets from both query sets. This approach requires only two random walks with restart, one from each of the query sets. The score derived from the set-based RWR from each of the query sets directly captures the proximity of a vertex \( v \in V \) to the set. More precisely, we modify above procedure for computing the scores as follows, where, without loss of generalization, we compute relatedness scores with respect to query set \( Q_1 \):

Similar to Personalized PageRank [21], for each \( q \in Q_1 \) we set the starting probability to \( s[q] = 1/|Q_1| \) and to \( s[v] = 0 \) for \( v \in V \setminus \{q\} \). As before, with probability \( \alpha \hat{M}[v, w] \), we walk from \( v \) to \( w \). With probability \( 1 - \alpha \), we jump back to a vertex in \( Q_1 \), chosen uniformly at random. To obtain the steady-state probabilities, we likewise iterate Equation 2 until convergence or for a predetermined number of iterations.

This procedure is performed for both \( Q_1 \) and \( Q_2 \), resulting in scores \( x_{Q_1}[v] \) and \( x_{Q_2}[v] \) for each vertex \( v \). The relationship center score (RC) of \( v \in V \) is the product of the two scores multiplied by a prior \( pr(v) \), derived from the overall importance or prominence of \( v \) (e.g., its PageRank score in \( \hat{M} \), popularity determined from external sources, etc):

\[ \text{RC}(v) = x_{Q_1}[v] \cdot x_{Q_2}[v] \cdot pr(v). \]  

Equation (7)
With this approach for vertex scoring, we can restrict the computational effort to two RWG processes in order to compute the scores. This makes the computation time independent of the input query sizes. In the experimental evaluation of this paper, we show that for the considered queries the quality of results is comparable to the CePS2-OR (Eq. 6) approach, while leading to a substantial speed-up of score computation.

As a final remark, recall that among the best vertices according to above score, we aim to select vertices that help explain the relationship. The types associated with the individual vertices can be very helpful to distinguish informative, high-scoring vertices from other, more generic high-scoring vertices. For knowledge graphs containing facts about real-world entities (such as the Espresso Knowledge Graph) we found that limiting relationship center candidates to entities that are (i) relatively prominent and (ii) of type event is an effective strategy that works well over a diverse range of queries. However, entity types of informative relationship centers could also be derived in a more principled way, i.e., by measures like mutual information between query entity types and possible candidate types.

5. RELATEDNESS CORES

In the previous sections we have discussed the identification of relationship centers and the ensuing connection to the query entities. Now, we discuss how the relationship centers can be expanded further to provide better relationship explanations. To this end, we use a three-phase algorithm to construct a coherent, yet compact subgraph around a relationship center $c$. We describe each phase of the algorithm in the rest of this section.

Phase 1: Key Entity Discovery. We compile a set of entities from the neighborhood of $c$, based on the relationship strength with $c$, e.g., as measured by the underlying entity-entity relatedness measure. For example, when starting with the event 2013 mass surveillance disclosures as center entity $c$, we would like to add entities like Edward Snowden, Glenn Greenwald, etc.

Starting with the initial set $V_c = \{c\}$ comprising just the center vertex, the algorithm works by iteratively adding the adjacent entity that is most related to $c$, based on the entity-entity similarity score measure introduced in Section 4, or on the degree-of-involvement scores, $x_v$. This score is computed via an additional random walk with restart, this time from the relationship center, $c$. This results in a score assigned to each vertex $v$ — denoted by $x_v(v)$. If the relationship center in a real-world knowledge graph is an actual event, the score $x_v[q]$ of a query entity $q$ can be interpreted as the degree of involvement of the entity in the event. The complete procedure is repeated until the set $V_c$ is sufficiently large, say $\gamma B'/2$ vertices, where $B'$ is the bound on the size of a relatedness core (the user-specified size constraint per core minus the number of vertices used to establish connections to the query sets) and the parameter $\gamma \in [0, 1]$ controls the fraction of key and context entities versus query context entities in a core.

Phase 2: Center Context Entity Generation.

The set of entities compiled in Phase 1 is further extended with entities that fill in the context necessary for explanation. Entities that should be added in this step typically have a broader scope and less specific relatedness, in the sense that they are also involved in many other events, e.g., United States, The Guardian, etc.

Given the current set of key entities, $V_c$, the goal of the second phase is the generation of additional context by adding generally important entities that are related to many of the key entities in $V_c$ but are not necessarily specific to $c$. Our algorithm to this end computes a dense subgraph around $V_c$, using the following greedy procedure: First, we mark each vertex $v \in V_c$ currently contained in the solution after key entity discovery as a terminal that must remain in the resulting solution. Second, we add all entities adjacent to the center $c \in V_c$ to an initial graph structure $G_c = (C_c, E_c)$, with

$$C_c = V_c \cup \{v \in V \mid (c, v) \in E\},$$

and

$$E_c = \{(v, w) \in E \mid v, w \in C_c\}.$$  

Based on the idea of computing dense subgraphs by iterative removal of low-weight vertices [10, 36], we expand the relatedness core as follows. We identify the non-terminal vertex $\hat{v}$ in $C_c$ with the smallest weighted degree with respect to the key entities:

$$\hat{v} = \arg \min_{v \in C_c \setminus V_c} \sum_{w \in V_c \setminus \{c, v\}} \text{sim}(v, w) \cdot x_v(w).$$

The vertex $\hat{v}$ is deleted from the graph $G_c$, and the procedure is repeated until the size constraint of $\gamma B'$ vertices is satisfied. The resulting subgraph thus consists of $\lfloor \gamma B' \rfloor$ key and $\lfloor \gamma B' \rfloor$ context entities.

Phase 3: Query Context Generation. In addition, we finally add entities that are highly related both to the relationship center as well as to a part of the query entities. Continuing above example, if we assume that New Zealand is a query entity, an appropriate query context entity is given by Five Eyes, the alliance of intelligence agencies involving the United States, United Kingdom, Australia, Canada and New Zealand, since this entity is highly related both to the central entity as well as to the query entity New Zealand.

Given the expanded relatedness core, the goal of the third phase is the addition of query context entities should give further insight into the relationship of the center entity to the query vertices by highlighting certain aspects of the relationship center that explain the involvement of the query entities. This is especially important for the frequently encountered case where some query entities are already directly connected to the relationship center, and no indirect connections have been added. For this purpose, we add the best $B' - \lfloor \gamma B' \rfloor$ vertices according to the score $x_{Q \cup \hat{V}}(v) x_v(v) \cdot \mu(v)$, i.e., additional, prominent vertices that are highly related both to the query entities as well as to the relationship center.

6. USING TEMPORAL INFORMATION

In our model so far, entity-entity relatedness was time-invariant. Here we extend our model by considering the temporal dimension as an additional asset for ensuring the coherence of relatedness cores. The entities in a core are often centered around a key event that connects the two input sets (e.g., the 2013 mass surveillance disclosures, connecting US politicians and European politicians). We want to ensure that the core is temporally coherent in the sense that many participating entities are relevant as of the time of the key event.

With each vertex $v \in V$ in the knowledge graph, we associate an activity function that captures the importance of (or public interest in) an entity as a function of time: $\alpha_v : T \rightarrow \mathbb{R}$. One way of estimating the values of this function is to analyze the edit history of Wikipedia: the more edits take place for an article in a certain time interval, the higher the value of the activity function. Other kinds of estimators may tap into longitudinal corpora that spans decades or centuries such as news archives or books. Our implementation is based on Wikipedia page view statistics. This entails that the estimated activity function is a discrete time-series, but we use the notation of continuous functions for convenience.

To make different entities comparable, we normalize the activity function of an entity as follows:

Definition 6.1 (Normalized Activity Function). Let $v \in V$ denote an entity with activity function $\alpha_v : T \rightarrow \mathbb{R}$. The normalized activity function of $v$ is defined as

$$A_v(t) = \frac{\alpha_v(t) - \mu_{\alpha_v}}{\sigma_{\alpha_v}},$$
with \( \mu_{\alpha_v} = \mathbb{E}[\alpha_v] \) and \( \sigma_{\alpha_v} = \sqrt{\mathbb{E}[(\alpha_v - \mu_{\alpha_v})^2]} \).

Thus \( A_v \) captures, for every time point, the number of standard deviations from the mean activity of \( v \). A similar technique is proposed by [14] based on the time-dependent connection discovery for co-buzzing entities.

To assess whether two entities are temporally coherent, we compare their activity functions. It turns out that many entities exhibit very pronounced peaks of activity at certain points. These peaks are highly characteristic for an entity. Therefore, we specifically devise a form of temporal peak coherence for comparing entities.

\[ tpc(x, y, T) = \int_{t_1}^{t_2} \max \{\min(A_x(t), A_y(t)) - \theta, 0\} \, dt \quad (12) \]

where \( \theta \) is a thresholding parameter to avoid over-interpreting low and noisy values.

In this definition, the time interval \( T \) can be set in various ways: the overall timespan covered by the knowledge graph, the overlap of the lifespans of the two entities (relevant for people or organizations), or the temporal focus specified by the user in the query that determines the two input sets \( Q_1 \) and \( Q_2 \).

Temporal Peak Coherence for Relatedness Cores. We harness the notion of temporal peak coherence in the algorithm for expanding relationship centers into relatedness cores. Specifically, when computing key entities surrounding the relationship center \( c \), we can enforce that only entities \( v \) are chosen with temporal coherence \( tpc_{2}(c, v) \) above a (tunable) threshold \( \tau \).

7. RELATED WORK

Explaining Relationships between Entities. Knowledge discovery in large graphs has been studied along several dimensions. Most relevant to this work is the extraction of subgraphs that explain the relationship between two or more input entities. [15] proposed the notion of connection subgraphs. Given an edge-weighted graph, a pair of query vertices, \((s, t)\), and a budget \( b \), the goal is to extract a connected subgraph containing \( s \) and \( t \) and at most \( b \) other vertices with a maximum value of a specified goodness function.

[38] later generalized this model to the case of a query vertex set \( Q \) with \(|Q| \geq 2\). In this approach, coined centerpiece subgraphs (CePS), the vertices in the graph are assigned a score based on random walks with restart from the query vertices. [34] proposed a connection subgraph discovery algorithm over RDF graphs, based on an edge-weighting scheme taking into account the edge-semantics of the respective RDF schema. [30] addressed the extraction of informative subgraphs with respect to a set of query vertices in the context of entity-relationship graphs. This approach is based on first computing a Steiner tree and then expanding it within a given budget. [12] partitioned graphs into communities that define the context of a vertex. Then, a subgraph is computed that connects all query vertices, aiming for strong connections at the intra-community and the inter-community levels. The methods outlined above have been used for improved graph visualization [11] and interactive graph mining [35].

Recently, [2] considered a similar problem, where, given a set of vertices, simple pathways connecting these are found, which are used to partition the input vertices into clusters of related vertices. [16] considered entity-relationship graphs in order to explain the relationship between pairs of individual entities that co-occur in queries. The algorithm first enumerates subgraph patterns by combining paths between the two query entities, and then ranks patterns based on interestingness measure derived from pattern instances.

Keyword Search over Graphs. Another area of related work is keyword search over (semi-)structured and relational data. In this setting, each vertex of a graph (e.g. derived from foreign-key relationships in a database) is associated with a textual description. The result of a query \( Q = \{t_1, t_2, \ldots, t_n\} \), consisting of several terms (keywords) \( t_i \), is a cost-minimal tree \( T \) (e.g., a Steiner tree based on edge weights) connecting a set of vertices such that for every term \( t \) in the query at least one of the vertices in \( T \) is a match for term \( t \). This field has been studied extensively [1, 8, 22, 26, 27, 28, 29], most prominent results being the methods/systems DISCOVER [26, 27], DBXplorer [1], BANKS [8, 28], and BLINKS [22]. The combination of keyword search over graphs and relationship analysis has been addressed by [31]. [13] give an overview and experimental comparisons for this research area.

Dense Subgraph Mining. Here the goal is to identify a set of vertices \( S \subseteq V \) maximizing a measure of density. A widely used notion of density is the average-degree, given by \( \frac{2|E(S)|}{|S|} \), where \( E(S) \) the set of edges contained in the spanning subgraph of \( S \). In the unconstrained case, computing the densest subgraph is polynomially solvable using a max-flow technique [20]. With cardinality constraints, the problem becomes NP-hard, though [32] gives an overview over this area. A greedy approximation algorithm computing the densest subgraph with a given number of vertices was proposed by [5]. [10] developed an \((1/2)\)-approximation algorithm. [3] studied further variants of the problem. [36] addressed the community-search problem with the goal of extracting dense components including a set of query vertices. [7] studied the dense subgraph problem in the streaming as well as in the MapReduce model. [40] investigate an alternative notion of density based on subgraph diameter.

Event Identification. In some sense, the extraction of relatedness cores resembles event detection over graph data. In this direction, [14] studied the discovery of events on the time axis (e.g., over query logs or news streams), but did not consider relationships between entities. Instead, the major asset for detecting events is the intensity of entity co-occurrence during certain time windows. In contrast to Espresso, this approach is not driven by given user query but aims to find all events in a real-time manner. With a similar focus on real-time discovery, [4] combined dense subgraph mining with a temporal analysis. Underlying is a graph of entities where edge weights are dynamically updated based on co-occurrence patterns in news or social-media streams. In this setting, events (or “stories”) correspond to dense subgraphs – continuously maintained as content items – that refer to multiple entities. More remotely related, [25] studied the problem of storytelling in entity-relationship graphs, where directed chains between target entities are extracted.

8. EXPERIMENTAL EVALUATION

This section presents experimental comparisons of Espresso with different baseline methods at two levels: (i) computing relationship centers, and (ii) end-to-end evaluations comparing the full explanation for the relationship between two entity sets. We discuss both the informativeness of the outputs, as judged in user studies, and the efficiency of the computation, as determined in run-time measurements. All experiments were conducted on a Dell PowerEdge M610 server, equipped with two Intel Xeon E5530 CPUs, 48 GB of main memory, and running Debian Linux (kernel 3.10.40.1-amd64-amp) as operating system. All algorithms have been implemented in C++11 (GCC 4.7.2).

Data. As previously described in Section 2, for the evaluation of our approaches we have extracted a large entity-relationship graph from the intersection of the YAGO2 and FreeBase knowledge bases, comprising roughly 3.7 million distinct entities, corresponding to Wikipedia articles. Using the links between Wikipedia pages, we have ex-
tracted an undirected graph by mapping each link from page \( u \) to page \( v \) to an undirected edge, \((u, v)\) for the corresponding entities. The resulting graph structure comprises almost 60 million edges. Wherever YAGO2 knows specific relationship labels between two entities, these are taken as edge labels for the knowledge graph, in addition to the generic label relatedTo for all edges.

**Queries.** We have manually created a set of 15 queries, each consisting of two sets of entities, \( Q_1, Q_2 \). There are 5 queries from each of the three categories Politicians (e.g., Heads of State (North America) – Heads of State (EU Countries), Japanese Heads of State – Chinese counterparts), etc., Organizations (e.g., Peace Organizations – Countries in Asia, Terrorist Organizations – North American and European Countries, etc.), and Countries (e.g., Eurozone Countries – Investment Banks, Russia – Asian Countries etc.). We have omitted the complete listing of queries used in our evaluation due to lack of space.

**Evaluation Metrics.** As all methods for identification of relationship centers yield ranked results, we evaluate the informativeness of the competitors by two metrics from information retrieval: precision and normalized discounted cumulative gain (NDCG). For this purpose, we have conducted a user study, in which a total of six judges assessed the relevance of the top-outputs (best entities of type event (manually identified) as relationship centers) for each of the 15 different test queries. The task of each judge was to evaluate the relevance of the output event with respect to the two input sets \( Q_1, Q_2 \), using a graded relevance scheme with values 0 (not relevant) to 3 (very relevant). The grades of the six judges were averaged for each test case. Each item was evaluated by all six evaluators. To compute precision, we mapped average grades from \([0, 1.5]\) onto 0 (irrelevant) and from \([1.5, 3]\) to 1 (relevant). The precision at rank \( k \) then is the fraction of relevant items among the first \( k \) results in the ranked list \( R = (r_1, r_2, \ldots, r_k) \):

\[
\text{Prec}_{@k}(R) = \frac{|\{r_i \in R \mid i \leq k, \text{rel}(r_i) \geq 1.5\}|}{k}.
\]

The evaluation by NDCG uses the average grades assigned by the judges on the full scale from 0 to 3. To compute the normalization for NDCG, we used a pooling approach: for each query, the results returned by all considered algorithms and settings are combined into a single list and ranked with respect to the relevance grades assigned by the human annotators. This list is denoted by \( C \). We compare the top-\( k \) result lists from the competing algorithms against the respective ideal ranking of the combined results. Here, ideal ranking means that results are sorted in descending order of the judges’ average grades. For ranking \( R = (r_1, r_2, \ldots, r_k) \) we compute

\[
\text{DCG}_{@k}(R) = \sum_{i=1}^{k} \frac{\text{rel}(r_i) - 1}{\log_{2}(i+1)},
\]

where, rel, denotes the average grade of the \( i \)-th item in \( R \), obtained in the user study. We finally normalize by the ideal ranking \( C \), to obtain

\[
n\text{DCG}_{@k}(R) = \frac{\text{DCG}_{@k}(R)}{\text{DCG}_{@k}(C)}.
\]

### 8.1 Extracting Relationship Centers

**Baseline: CePS.** We compare our method for extracting relationship centers against the extension of the centerpiece sub-graph algorithm by [38] for sets, considering both the CePS-AND (cf. Equation 5) as well as the CePS2-OR (cf. Equation 6) variant. Recall that this baseline performs random walks with restarts from all query nodes \( q_i \in Q \) given a single node set. These approaches are abbreviated as CePS-AND and CePS2-OR, respectively.

**Informativeness.** We evaluate the relevance of computed relationship centers by Espresso and CePS by using the plain top-5 ranking (based only on random walk scores) and the top-5 after reranking.

In this setting, the reranking procedure multiplies the relatedness score with the entity prior (as described in Equation 7), which, in this experiment, corresponds to the square root of the relative amount of Wikipedia pageviews for the corresponding article over the time period from January 1, 2013 to July 31, 2014. A total number of six judges responded to the study, which comprises 249 distinct (question, event) pairs. The results of comparing the relationship centers of our set-based random walk algorithm described in Section 4 – denoted by RC – against those by the extension of CePS for query sets are shown in Table 1, for the top-5 results.

The explicit distinction between the two input sets of entities, \( Q_1, Q_2 \) benefits Espresso and CePS2-OR, whereas CePS-AND requires central vertices relevant to all query entities. For 6 of the 12 settings (area, relatedness measure, reranking), CePS2-OR is slightly better than Espresso (RC), for 3 queries both achieve the same precision, and Espresso even produces the best result for 3 queries. Over all queries, Espresso achieves average precision of 0.75, whereas CePS2-OR and CePS-AND achieve average precision of 0.77 and 0.58, respectively. Thus, Espresso manages to attain 97% of the precision of CePS2-OR, while only requiring two RWRs from the query sets. The results obtained by using Milne-Witten inlink overlap (MW) as relatedness compare favorably with the results obtained from employing the KORE measure. Reranking based on entity popularity improves the result quality significantly for the MW measure. Thus, the combination of MW and popularity-based reranking is used as the default strategy for Espresso and CePS.

The NDCG results for the two algorithms, RC and the CePS variants are shown in Table 2. Overall, Espresso achieved average NDCG@5 of 0.67, while CePS2-OR and CePS-AND achieved scores of 0.68 and 0.55, respectively. Espresso thus retains 98% of the score.

**Run-Time Efficiency.** From an asymptotic complexity perspective, the analysis is straightforward:

- The CePS variants require a random walk process from each of the \(|Q_1| + |Q_2|\) query vertices. Each RWR iteration has a time complexity of \(O(|E|)\). In our implementation we use a fixed number of \( I \) iterations. Thus, the time complexity for this method is \(\Theta\left((|Q_1| + |Q_2|)I|E|\right)\).

- For our random walk approach RC described in Section 4, we conduct one random walk with restart from each query set. With a fixed number of \( I \) iterations, this procedure results in a time complexity of \(\Theta(2I|E|)\).

It is evident that the time for computing RC scores is independent of the input set sizes and only depends on the size of the overall graph, whereas the CePS method has linear dependence on the size of the input entity set(s). Due to lack of space, we skip further details of runtime evaluation of center computation algorithms, instead present the end-to-end performance of all three relationship explanation techniques in the next section.

### 8.2 Relationship Explanation

We conducted end-to-end experiments for generating relationship explanations for each of our 15 test queries, comparing Espresso to an extension of the recently proposed Rex approach of [16] to sets of input entities rather than individual pairs, as well as the CePS algorithm.

**Baseline: REX.** The Rex algorithm in its original form computes relationship explanations in the form of instances of informative graph patterns (edge-labeled templates) that connect two nodes. Due to its restriction to a single pair of query nodes we extend Rex in the following way: Given input sets \( Q_1, Q_2 \) of entities, we choose \( K \) pairs of query entities \((q_1, q_2)\) \(\in Q_1 \times Q_2\), \(1 \leq i \leq K\) and aggregate the individual results into a combined result. Specifically, this involves three steps: (1) select query entity pairs, (2) for each \((q_1, q_2)\) pair
compute the best \(p \) patterns \(P_1, \ldots, P_p \) (for pattern ranking we use a combination of the size and monochrom measures described in the original paper [16]), and (3) add the heaviest (in terms of sum of edge weights) \(v \) instances of each of the \(p \) best patterns to the candidate subgraph \(C = (V_c, E_c) \). Subsequently, in order to satisfy the size constraint on the output, while maintaining a coherent relationship explanation, we prune the candidate subgraph as follows. We arrange the vertices in \(C \) that do not correspond to query entities in a min-heap based on the weighted degree in \(C \). As long as the cardinality constraint is violated, we repeatedly identify a vertex for pruning. For this purpose we select the vertex from the heap with lowest weighted degree that does not disconnect \(C \), in the sense that each inner vertex \(v_i \in V_c \setminus (Q_1 \cup Q_2) \) remains connected to at least one vertex in both \(Q_1 \) and \(Q_2 \). If no such vertex exists, we simply prune the inner vertex \(v_i \) with lowest weighted degree, potentially having to prune dangling vertices (i.e. vertices with degree 1 after deletion of \(v_i \) ) afterwards. This way, we aim to obtain a coherent, dense graph that maintains as much informativeness as possible.

We used the following parameters for the extended Rex algorithm. The pattern size was restricted to 4. For each query we use up to 20 pairs of query entities \((q_1, q_2) \in Q_1 \times Q_2 \) to compute pairwise explanations. If it holds \(|Q_1||Q_2| > 20 \), we randomly select 20 pairs. Our implementation uses bidirectional expansion from the pair of query vertices to generate the path patterns. In order to cope with the combinatorial explosion, we restricted the number of generated paths per pair to 20,000. We have implemented the PathUnionBasic algorithm to generate the patterns, as described by [16]. For each pair of query entities we compute the 5 best patterns as measured by size (smaller better) and monochrom (larger better). Finally, for each of the patterns, we add the heaviest two instances to the candidate subgraph, followed by the pruning phase to consolidate a small and dense connecting subgraph. The algorithm was implemented in C++ and compiled at the highest available optimization level. As input we used the Espresso Knowledge Graph with MW-weighted edges, and add additional directed, labeled edges for YAGO2 facts between the entities.

Baseline: CePS. As second baseline, we implemented the entire algorithm for centerpiece subgraph (CePS) extraction, as described by [38], and compute results in the CePS2-OR setting, i.e. random walks with restarts from all query entities with the aggregation based on Equation 6, followed by the EXTRACT algorithm to connect the best query entity from either set to the centerpieces vertices (via a so-called key path) in a dynamic programming fashion. Regarding the dynamic programming procedure to extract key paths, in our implementation we will directly connect a query entity to the current center if the latter is already present in the partially built graph and a direct edge exists, rather than extracting a key path. The input graph used is the Espresso Knowledge Graph with MW-weighted edges. Since our previous user study indicated the combination of MW-weights with popularity based reranking as entity prior to be the superior strategy, we incorporate the reranking as follows. After all random walk processes have finished, we multiply the resulting aggregated score for each individual vertex with the entity prior (square-rooted relative popularity). In addition, after the aggregated score has been computed, we also multiply the scores obtained by each individual random walk process with the entity prior. We implemented CePS in C++, compiled using the highest available optimization level.

The cardinality constraint we used in this experiment was 6, for all three algorithms (Rex, CePS, Espresso). For the random walks in CePS and Espresso we use 5 iterations and set the restart probability to 0.5. For Espresso, the parameters we specify are a number of 3 relationship centers to use, fraction of key and context entities vs. query context entities set to \(\gamma = 0 \). This represents the first stage visualization of the proposed Espresso system, where we rather show more yet compact events, to give a good overview. In a user-facing system, the individual events would then be shown with more key/context and query-context when selected by the user (leading to recomputation with a higher value for \(\gamma \) and cardinality constraint). For all queries, there was at least one query entity from either query set directly connected to an event. As a result, it was not required to compute the connections to the query entities, rather we only had to connect the

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Relatedness</th>
<th>Renark</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Avg</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
<th>Avg</th>
<th>Q11</th>
<th>Q12</th>
<th>Q13</th>
<th>Q14</th>
<th>Q15</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC</td>
<td>KORE</td>
<td>✓</td>
<td>1.00</td>
<td>0.40</td>
<td>0.60</td>
<td>1.00</td>
<td>0.60</td>
<td>0.72</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>0.60</td>
<td>0.80</td>
<td>0.80</td>
<td>0.65</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.44</td>
</tr>
<tr>
<td>RC</td>
<td>KORE</td>
<td>✗</td>
<td>1.00</td>
<td>0.80</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>0.92</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>1.00</td>
<td>0.92</td>
<td>1.00</td>
<td>0.40</td>
<td>0.20</td>
<td>0.40</td>
<td>0.80</td>
<td>0.56</td>
<td></td>
</tr>
<tr>
<td>RC</td>
<td>MW</td>
<td>✓</td>
<td>1.00</td>
<td>1.00</td>
<td>0.80</td>
<td>0.80</td>
<td>1.00</td>
<td>0.92</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.92</td>
<td>1.00</td>
<td>1.00</td>
<td>0.20</td>
<td>0.60</td>
<td>0.60</td>
<td>0.68</td>
<td></td>
</tr>
<tr>
<td>RC</td>
<td>MW</td>
<td>✗</td>
<td>1.00</td>
<td>1.00</td>
<td>0.80</td>
<td>0.80</td>
<td>1.00</td>
<td>0.92</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.92</td>
<td>1.00</td>
<td>1.00</td>
<td>0.20</td>
<td>0.60</td>
<td>0.60</td>
<td>0.80</td>
<td></td>
</tr>
<tr>
<td>CePS SETS-OR</td>
<td>KORE</td>
<td>✓</td>
<td>1.00</td>
<td>0.40</td>
<td>0.60</td>
<td>1.00</td>
<td>0.80</td>
<td>0.76</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>0.80</td>
<td>0.84</td>
<td>0.60</td>
<td>0.40</td>
<td>0.60</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td></td>
</tr>
<tr>
<td>CePS SETS-OR</td>
<td>KORE</td>
<td>✗</td>
<td>1.00</td>
<td>0.80</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>0.92</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.88</td>
<td>0.88</td>
<td>1.00</td>
<td>0.40</td>
<td>0.20</td>
<td>0.40</td>
<td>0.40</td>
<td>0.48</td>
</tr>
<tr>
<td>CePS SETS-OR</td>
<td>MW</td>
<td>✓</td>
<td>1.00</td>
<td>1.00</td>
<td>0.80</td>
<td>0.80</td>
<td>1.00</td>
<td>0.92</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.88</td>
<td>1.00</td>
<td>1.00</td>
<td>0.40</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
</tr>
<tr>
<td>CePS SETS-OR</td>
<td>MW</td>
<td>✗</td>
<td>1.00</td>
<td>1.00</td>
<td>0.80</td>
<td>0.80</td>
<td>1.00</td>
<td>0.92</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.88</td>
<td>1.00</td>
<td>1.00</td>
<td>0.40</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
</tr>
<tr>
<td>CePS SETS-AND</td>
<td>KORE</td>
<td>✓</td>
<td>0.80</td>
<td>0.80</td>
<td>0.40</td>
<td>0.80</td>
<td>0.60</td>
<td>0.68</td>
<td>0.80</td>
<td>0.80</td>
<td>1.00</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
<td>0.20</td>
<td>0.40</td>
<td>0.40</td>
<td>0.00</td>
<td>0.28</td>
</tr>
<tr>
<td>CePS SETS-AND</td>
<td>KORE</td>
<td>✗</td>
<td>1.00</td>
<td>0.80</td>
<td>0.40</td>
<td>0.80</td>
<td>0.60</td>
<td>0.72</td>
<td>0.80</td>
<td>0.80</td>
<td>1.00</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
<td>0.20</td>
<td>0.40</td>
<td>0.40</td>
<td>0.00</td>
<td>0.28</td>
</tr>
<tr>
<td>CePS SETS-AND</td>
<td>MW</td>
<td>✓</td>
<td>1.00</td>
<td>1.00</td>
<td>0.60</td>
<td>0.60</td>
<td>1.00</td>
<td>0.84</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>0.80</td>
<td>1.00</td>
<td>0.40</td>
<td>0.60</td>
<td>0.60</td>
<td>1.00</td>
<td>0.80</td>
</tr>
<tr>
<td>CePS SETS-AND</td>
<td>MW</td>
<td>✗</td>
<td>1.00</td>
<td>1.00</td>
<td>0.60</td>
<td>0.60</td>
<td>1.00</td>
<td>0.84</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>0.80</td>
<td>1.00</td>
<td>0.40</td>
<td>0.60</td>
<td>0.60</td>
<td>1.00</td>
<td>0.80</td>
</tr>
</tbody>
</table>

Table 1: Prec@5 over the set of 15 test queries
event with the appropriate query entities and could add one query context entity to each of the relationship centers. As discussed previously, one approach to ensure informativeness of the central entities is to ensure a certain level of prominence of the considered events. In CePS, where we do not filter by entity types, the prominence of the center entities is ensured by the popularity-based reranking step. In the Espresso algorithm, we discard event entities from consideration with less than 50 daily pageviews on average, and a maximum number of daily pageviews of less than 500, prior to reranking. As a result, out of the 89,321 entities originally marked as events in the Espresso Knowledge Graph, a total number 7,056 events remains as candidates for relationship centers.

Both CePS as well as Espresso include the subgraph induced by the query and center entities, i.e. all edges appearing between any pair of vertices in the computed solution, including edges within the query sets. The labels displayed are the most specific ones associated with each included edge for CePS and Espresso, and the labels contained in the best patterns for Rex. We omit the generic label relatedTo in the displayed solutions.

**Informativeness.** The resulting explanations were assessed by human judges. For every query, the output of the three algorithms Espresso, extended Rex, as well as CePS2-OR was presented pairwise to the users, who had to choose whether (i) the first graph provides a better explanation, (ii) both graphs are equally good or (iii) the second graph provides a better explanation. For each of the 15 queries the users evaluated all 3 possible pairings (Rex vs CePS2-OR, CePS2-OR vs Espresso, Espresso vs Rex). We used the same graph layout for all three algorithms. A total number of six judges responded to the study. We show the resulting judgements in Tables 3(a)-(c). It becomes clear that the judges clearly favor the graphs computed by Espresso over the two baseline solutions. In 82% of all considered cases, the solution returned by Espresso was favored over the solution returned by Rex, while both were rated equally good in 10% of all cases. When compared with CePS, Espresso returned the better solution in 61% of all cases, and both were evaluated equally good in 22% of all cases. The comparison between CePS and Rex shows that both were evaluated equally good in 22% of the considered cases, while CePS computed the better solution for 57% of the queries.

<table>
<thead>
<tr>
<th></th>
<th>Rex better</th>
<th>both equal</th>
<th>Espresso better</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(a) Rex vs. Espresso</strong></td>
<td>7 (8%)</td>
<td>9 (10%)</td>
<td>74 (82%)</td>
<td>90</td>
</tr>
<tr>
<td><strong>CePS2-OR better</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CePS2-OR better</td>
<td>14 (16%)</td>
<td>21 (23%)</td>
<td>55 (61%)</td>
<td>90</td>
</tr>
</tbody>
</table>

Table 3: End-To-End User Study Results

**Efficiency.** Regarding the efficiency, we compare the end-to-end running times of the considered algorithms in Figure 2. Espresso is the fastest of all three algorithms, and exhibits execution times always below 23 seconds over all queries, highlighting its independence from the size of the query sets. In general, the number of random walks executed from the relationship centers differs for the different questions, since sometimes the extracted cores overlap, leaving budget for additional relationship centers. Thus, for 9 of 15 queries the running time of Espresso amounts to ~16 seconds, for 4 queries to ~19 seconds, and for one query (Q3) to 23 seconds.

On the other hand, the running time of CePS is directly dependent on the size of the query sets, since it executes a random walk from each query entity. The overall running times range from 6 seconds (Q12) to 123 seconds (Q7), while CePS is slower than Espresso for 12 of the 15 queries. The time required to compute the relationship explana-
We can thus reason, that in the knowledge graph scenarios, it is very important to enforce informativeness of the central entities. In the experimental evaluation of Espresso, this is ensured by focusing on central vertices of type event, and extracting coherent cores by combining the relatedness center scores w.r.t. to the query entities with relatedness to the center event, in order to identify good query context entities.

The restriction to entities typed event gives an advantage to Espresso, since many non-informative entities are not considered as central vertices (e.g. for the case of countries in the two query sets, we can expect other related/bordering countries to be assigned high scores – however, connections via such vertices are hardly informative and do not give much insight into the relationship). The main insight is thus, that entity type restriction is a crucial step for a good relationship explanation. It should further be mentioned, that for the settings of Espresso with core cardinality $B = 1$, and CePS in the CePS2-SETOR setting employing an event-filtering step similar to Espresso, we would expect very similar results. Espresso can however support further analysis by expanding the identified events into coherent explanations via the addition of key, context, and query context entities. It remains a challenging problem to automatically identify appropriate entity types, to distinguish between potentially informative and less informative central vertices. For many queries involving real-world entities, the restriction to events works very well. This advantage carries a certain risk however, if good connections are missed because no event exists that involves the query entities (but, rather, subsets of the two query sets are for example members of the same organization, etc.). One strength of Espresso lies in its flexibility to adapt to a wide range of user specifications such as using temporal coherence to detect key entities.

9. SUMMARY

Entities and their relationships play an increasingly important role as semantic features for numerous kinds of search, recommendation, and analytics tasks – on the Web, for social media, and in enterprises. Explaining relationships between entities and between sets of entities is a key issue within this bigger picture. The Espresso framework presented in this paper is a novel contribution along these lines. Our experiments, including user studies, demonstrate the practical benefits of Espresso. Future technical work include the automatic derivation of informative entity types based on the query as well as harnessing our methods for better summarization of online contents, for both individual documents (e.g., articles in news or magazines) and entire corpora (e.g., entire discussion threads, entire forums, or even one day’s or one week’s entire news). Users are overwhelmed with content; explaining and summarizing content at a cognitively higher level is of great value.

10. REFERENCES