# Meta Structures in Knowledge Graphs

Valeria Fionda[1] and Giuseppe Pirrò[2]

[1] DeMaCS, University of Calabria, Italy
fionda@mat.unical.it
[2] Institute for High Performance Computing and Networking, ICAR-CNR, Italy
pirro@icar.cnr.it

**Abstract.** This paper investigates meta structures, schema-level graphs that abstract connectivity information among a set of entities in a knowledge graph. Meta structures are useful in a variety of knowledge discovery tasks ranging from relatedness explanation to data retrieval. We formalize the meta structure computation problem and devise efficient automata-based algorithms. We introduce a meta structure-based relevance measure, which can retrieve entities related to those in input. We implemented our machineries in a visual tool called MEKoNG. We report on an extensive experimental evaluation, which confirms the suitability of our proposal from both the efficiency and effectiveness point of view.

## 1 Introduction

Knowledge Graphs (KGs) are becoming a common support in many application domains including information retrieval, recommendation, clustering, entity resolution, and generic exploratory search. One fundamental task underpinning these applications is the extraction of connectivity structures such as paths or graphs between entities. At the data level, these structures reflect fine-grained semantic associations like: K. Knuth award Turing Award award$^{-1}$ John Hopcroft; the abstraction of these structures by using schema information (e.g., typing, domain and range) allows to capture meta information (e.g., Scientist award Prize award$^{-1}$ Scientist). Most of current efforts focus on finding simple connectivity structures like (meta) paths between *a pair* of entities [2, 17]. This has several limitations: (i) paths are not enough to capture complex relationships; (ii) limiting the input to a pair of entities does not allow to find refined associations both at the data and schema level; (iii) enumerating paths is a computationally hard problem. Recent approaches (e.g., [1]) focus on finding richer structures *only* but do not report on their usage in knowledge discovery tasks (see Section 5).

In this paper we focus on *meta structures*, schema-level graphs that abstract connectivity information among a set of entities in a knowledge graph. We study the problem of both finding *meta structures* and computing *meta structure-based relevance* and define: (a) efficient algorithms to isolate the subgraph connecting the input entities *without enumerating paths*; (b) techniques to pick the *most relevant portion of this subgraph*; (c) techniques to *abstract* data level information into a meta structure; (d) *relevance measures* based on meta structures; (f) *user supports*. The contributions of this paper are as follows:

- Automata-based algorithms to find a subgraph connecting a set of entities.
- Layered-Tuple-Relevance (LTR), a meta structure-based relevance measure.
- A visual tool called MEKoNG implementing our approach.
- An experimental evaluation, which shows the efficiency of our proposal both in terms of running time and in concrete knowledge discovery tasks.

The goal of this paper is on the efficient computation of meta structures and their usage for relevance computation; the effectiveness from the user point of view of several types of connectivity structures has been investigated in [17].

Meta structures are useful in a variety of tasks ranging from finding/visualizing connectivity among entities to recommender systems (e.g., by computing the relevance between items already purchased and new items). In Section 1.1, we describe an instantiation of our proposal in the MEKoNG tool, useful to both discover entity relatedness and recommend related entities; other applications of our framework (e.g., entity resolution) are considered in Section 4. The paper is organized as follows. Section 2 describes the problem and the algorithms. Meta structure-based relevance is discussed in Section 3. Experiments are discussed in Section 4. We review related work in Section 5 and conclude in Section 6.

### 1.1 Running Example

We now illustrate MEKoNG, a tool that leverages the low-level services provided by our framework. We consider the tuple (A. Aho, J. Hopcroft, D. Knuth) as input and focus on the following tasks: (i) retrieve and explore a meta structure and its instances; (ii) retrieve the top-5 relevant entities.

Fig. 1 (a) shows a meta structure retrieved for this entity tuple; we can see that it is a graph including three entities of type Scientist and two entities of type Award. In particular, one of the scientist has been a doctoral student of a second Scientist; note also that all three scientist share the same Award and that two of them also share a second award. The level of expressiveness of this meta structure goes beyond the expressiveness of its (meta) paths taken separately. In fact by using meta paths only it would not have been possible to capture constraints like the common Award. MEKoNG allows to explore a meta structure and its instances giving insights about the relatedness among the input entities. This is extremely useful in large KGs as it allows to find out previously unknown knowledge that is of relevance and understand how it is of relevance. We can see (Fig. 1 (b)) that the Award that the three Scientist share is the IEEE von Neumann Medal and that D. Knuth and J. Hopcroft share the Turing Award.

Building upon meta structures, MEKoNG allows to assess entity relevance. This occurs by replacing nodes in a meta structure with source entities and picking one of the remaining nodes as target. In the example (see Fig. 1 (c)) A. Aho and J. Hopcroft are used as seed entities thus replacing the leftmost Scientist nodes in Fig. 1 (a) and the target is the remaining Scientist. The top-5 more relevant entities, ranked according to the LTR relevance measure presented in Section 3, are shown in Fig. 1 (d). The top relevant result is T. Hoare followed by I. Sutherland. We can explain this ranking with the fact that LTR when using

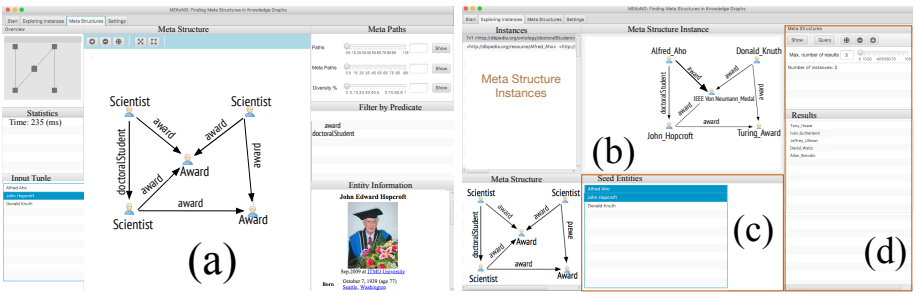


**Fig. 1.** The MEKoNG system.

the seed entities A. Aho and J. Hopcroft and the meta structure in Fig. 1 (a) can both discover Award entities and take into account their *specificity* (i.e., how many other scientists have a particular Award). I. Sutherland and T. Hoare are ranked higher than, for instance, J. Ullman that share more awards, because the former share the Turing Award with J. Hopcroft and this award is less common than the ACM Fellowship shared with J. Ullman.

## 2 Discovering Meta Structures

In this section we introduce our approach to find meta structures. We start with some preliminaries and formalize the problem in Section 2.1. Then, we introduce algorithms to find a meta structure instance for an input tuple in Section 2.2. In Section 2.3 we discuss how to abstract a meta structure instance.

### 2.1 Problem Formalization

A Knowledge Graph (KG) is a heterogeneous network where nodes are entities of different types and edges model different types of semantic relationships. Yago, DBpedia, and Freebase are a few examples of popular KGs available in the RDF standard data format. Due to the generality of our approach, in what follows we provide a general notation that models graph data. A KG is a directed node and edge labeled graph $G{=}(V, E)$ with two node mapping functions $\phi_i{:}V \rightarrow \mathcal{L}_v$, which assigns to each node a unique *id* and $\phi_t{:}V \rightarrow 2^{\mathcal{L}_s}$, which assigns to each node a set of types in $\mathcal{L}_s$. An edge mapping function $\varphi{:}E \rightarrow \mathcal{L}_e$ associates to each edge a type from $\mathcal{L}_e$. To structure knowledge, KGs resort to an underlying schema, which is defined in terms of entity *types* and their links.

**Definition** 1 (**Knowledge Graph Schema**). Given a KG G and its mapping functions $\phi_t{:}V \rightarrow 2^{\mathcal{L}_s}$ and $\varphi{:}E \rightarrow \mathcal{L}_e$, the schema $T_G$ of G is a directed graph defined over $\mathcal{L}_s$ and $\mathcal{L}_e$, that is, $T_G{=}(\mathcal{L}_s{,}\mathcal{L}_e)$.

An example of KG schema is reported in Fig. 2 (a); it allows to abstract and define data and their compatibility (e.g., *fellow* links Scientist and Association).

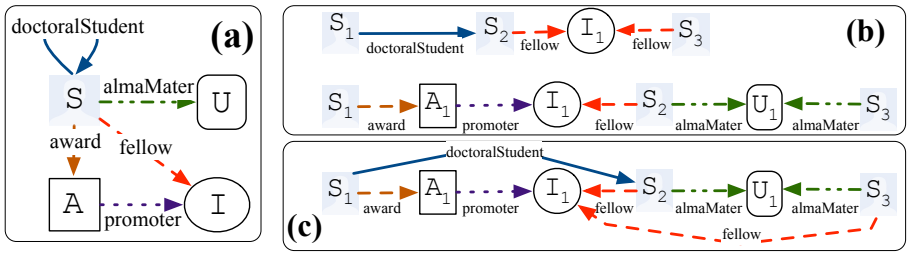


**Fig. 2.** A KG schema (a); some meta paths (b); a meta structure (c). Edge types/colors represent different kinds of relationships; node shapes represent different entity types.

Given a KG, a meta path is essentially an abstract data representation, which uses schema information. Examples of meta paths are shown in Fig. 2 (b). Meta paths can only capture simple relationship between entities, while meta structures, being modeled as graphs, allow to capture more complex relationships. As an example, the meta structure in Fig. 2 (c), can model the fact that the Association $\mathtt{I}_1$ *is shared* between the Scientists $S_3$, $S_2$, and the Award $A_1$. This aspect cannot be modeled by using the two meta paths in Fig. 2 (b). We now introduce the notion of $m$-meta structure, which generalizes the notion of meta structure, defined for a pair of entities [10], to a tuple of arbitrary length.

**Definition** 2 ($m$**-Meta Structure**). Given a KG schema $T_G{=}(\mathcal{L}_s, \mathcal{L}_e)$ and an entity tuple $t = \langle e_1, ..., e_m \rangle$, an $m$-meta structure for $t$ is a graph $S{=}(N, M, T_s)$, where $N{\subseteq}\mathcal{L}_s$ is a set of entity type nodes, $M$ a set of edges and $T_s{=}\langle T_1, ..., T_m\rangle{\subseteq}N$ is a set of entity types each corresponding to an input entity. For any edge $(u, v){\in}M$ we have that $(u, v){\in}\mathcal{L}_e$.

**Definition** 3 ($m$**-Meta Structure Instance**). An instance of an $m$-meta structure $S{=}(N, M, T_s)$ for $t = \langle e_1, ..., e_m \rangle$ on a KG $G$, is a subgraph $s{=}(N_s, M_s)$ of $G$ such that there exists a mapping for s, $h_s{:}N_s{\rightarrow}N$ satisfying the following constraints: (i) for any entity $v{\in}N_s$ its type $h_s(v){\in}\phi_t(v)$; (ii) for any edge $(u, v){\in}(\notin)M_s$ we have that $(h_s(u), h_s(v)){\in}(\notin)M$.

The first goal of this paper is to tackle the problem of computing an $m$-meta structure given a knowledge graph $G$, its schema $T_G$ and an input tuple $\langle e_1, ..., e_m \rangle$. This goal can be formalized via the following general problem:

| | |
|---|---|
| **Problem**: | $m$-MetaStructureComputation |
| **Input**: | A KG $G$, a KG schema $T_G$, an entity tuple $\langle e_1, ..., e_m \rangle$ |
| **Output**: | An $m$-meta structure $S$ |

To solve $m$-MetaStructureComputation we address two subproblems: **SP1**, which focuses on building an $m$-meta structure instance $s$ (Section 2.2); and, **SP2**, which is about abstracting $s$ by using the KG schema $T_G$ (Section 2.3).

### 2.2 SP1: Building $m$-Meta Structure Instances

In computing an $m$-meta structure instance for the input tuple $t = \langle e_1, ..., e_m \rangle$, our algorithm sets a horizon $h$; this parameter bounds the portion of the graph considered where entities in $t$ are connected. If one were not to limit the search horizon, then paths connecting entities in $t$ can potentially span over large portions of $G$. These generic paths would be not informative as they fail to capture the essential relationships between entities in $t$. Indeed, if a too large horizon is considered then the whole $G$ (more precisely, the connected component where entities in $t$ lie) can trivially become the sought $m$-meta structure instance.

In what follows, we refer to the problem of computing an $m$-meta structure instance that connects the $m$ entities in input as $m$-MetaStrInstComp. One way to approach the above problem could be to compute paths of length at most $h$ interlinking the entities in $t = \langle e_1, ..., e_m \rangle$ and then merging them to obtain the $m$-meta structure instance. Some existing approaches (e.g., [2, 17]) obtain paths via SPARQL queries and then merge (a subset of) them according to different strategies. From a computational point of view, materializing paths and merging them is not an efficient choice. This is because the number of paths can be exponential, thus requiring both exponential space (to store them) and exponential time (to iterate over them). In what follows we give an algorithm showing that $m$-MetaStrInstComp can be efficiently solved.

**Proposition 4** $m$-MetaStrInstComp can be solved in time $\mathcal{O}(m \times h \times |G|)$

To prove the above result, we provide an algorithm based on automata theory. The algorithm encodes the input entity tuple $t = \langle e_1, ..., e_m \rangle$ as a regular expression having the form $e_t$=(•)*/[$e_2$]/.../(•)*/[$e_{m-1}$]/(•)*/[$e_m$]; here, • is a wild-card, representing a generic edge label in $\mathcal{L}_e$ and the notation $[e_i]$ encodes a test, which checks whether an edge endpoint is equals to $e_i$, one of the entities in $t = \langle e_1, ..., e_m \rangle$; such kind of regular expression can be represented via a NFA [5] over the alphabet $\bigcup_{i=1}^{m} \{[e_i]\} \cup \{\bullet\}$ with state transitions occurring when finding an input entity. We refer to this automaton as *tuple automaton* $A_t$ (see Fig. 3). The set of strings obtained by concatenating the edge labels of paths passing through $e_1, ..., e_m$ are the set of strings generated by the language corresponding to $e_t$ and recognized by $A_t$.

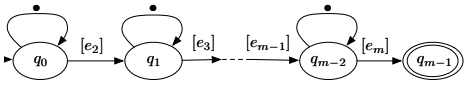


**Fig. 3.** Tuple-automaton.

Base Algorithm. We are now ready to present the algorithm to compute the $m$-meta structure instance linking entities in $t = \langle e_1, ..., e_m \rangle$. The algorithm includes two main steps: *(i)* building a directed label graph $G \times A_t$ (see Algorithm 1); *(ii)* filtering the portion of the input KG $G$ that should not be part of the $m$-meta structure instance identified in the first step (see Algorithm 3). The graph $G \times A_t$ is built via the procedure reported in Algorithm 1 that performs an optimized Breadth-First Search on the graph $G$, according to the automaton $A_t$.

**Input** : KG $G$, tuple $t$=$\langle e_1, ..., e_m \rangle$, horizon $h$
**Output**: $G \times A_t = (V', E')$ /* marking of $G$ with states of $A_t$ */
1: $A_t$=`buildTupleAutomaton`($t$) /* build the tuple automaton for the tuple $t$ */
2: $V'$=$\{(e_1, q_0, 0)\}$; $E' = \emptyset$ /* 0 is the starting depth and $q_0$ the initial state of $A_t$ */
3: $toVisit$=$visited$=$\{(e_1, q_0, 0)\}$
4: **while** $toVisit \neq \emptyset$ **do**
5: $(v, q_i, d)$= `extract`($toVisit$) /* remove the pair inserted first */
6: **for all** $\langle (v', q_j), p \rangle$ in `expandState`($(v, q_i)$,$G$,$A_t$) **do**
7: **if** $((v', q_j, d+1) \notin visited)$ and $(d<h)$ and $(|t|\text{-}j\text{-}1) <(h\text{-}d))$ **then**
8: $toVisit$.`add`$((v', q_j, d+1))$
9: $visited$.`add`$((v', q_j, d+1))$
10: $V'$=$V' \cup \{(v', q_j, d+1)\}$
11: $E'$=$E' \cup \{((v, q_i, d), p, (v', q_j, d+1))\}$
12: **return** $G \times A_t$

**Algorithm 1:** `buildMarkedGraph`($G$, $t$, $l$)

**Input** : node-state pair $(v, q)$, KG $G$, tuple-automaton $A_t$
**Output**: $\langle (node, state), edgeLabel \rangle$ pairs $L$
1: $L = \emptyset$
2: **for all** $(v, p, v') \in G$ **do**
3: **if** $(q, [v'], q') \in A_t$ **then**
4: $L$.`add`$(\langle (v', q'), p \rangle)$
**else** $L$.`add`$(\langle (v', q), p \rangle)$
5: **for all** $(v', p, v) \in G$ **do**
6: **if** $(q, [v'], q') \in A_t$ **then**
7: $L$.`add`$(\langle (v', q'), p^- \rangle)$
**else** $L$.`add`$(\langle (v', q), p^- \rangle)$
8: **return** $L$

**Algorithm 2:** `expandState`($(v, q)$, $G$, $A_t$).

**Input** : $G \times A_t$: Marking of $G$ with states of $A_t$
**Output**: $s = (N_s, M_s) \subseteq G$: $m$-meta structure Instance
1: $toVisit = visited = \{(e_m, q_{m-1}, k) \mid (e_m, q_{m-1}, k) \in G \times A_t\}$
2: $N_s = \emptyset$, $M_s = \emptyset$
3: **while** $toVisit \neq \emptyset$ **do**
4: $(v, q, d)$= `extract`($toVisit$)
5: $N_s$=$N_s \cup \{v\}$
6: **for all** $((v', q', d-1), p, (v, q, d))$ in $G \times A_t$ **do**
7: $N_s$=$N_s \cup \{v'\}$
8: $M_s$=$M_s \cup \{(v', p, v)\}$
9: **if** $(v', q', d-1) \notin visited$ **then**
10: $toVisit$.`add`$((v', q', d-1))$
11: $visited$.`add`$((v', q', d-1))$
12: **return** s

**Algorithm 3:** `filterMetaStructureInstance`($G \times A_t$)

The first step is the construction of the tuple-automaton $A_t = \langle Q, \Sigma, q_0, \{q_{m-1}\}, \delta \rangle$ associated to $e_t$ (line 1) and reported in Fig. 3; here, $Q$ is the set of states, $\Sigma$

is the alphabet, $q_0$ the initial state, $\{q_{m-1}\}$ is the set of final states, and $\delta$ the transition function. The size of $A_t$ linear in the size of the input tuple, that is, $|A_t| = \mathcal{O}(m)$. $A_t$ is used to build the labeled graph $G \times A_t$ whose nodes are a subset of $V \times Q \times \{0, ..., h\}$. $G \times A_t$ contains an edge from the node $(v, q, d)$ to the node $(v', q', d+1)$ labeled with $p \in \mathcal{L}_e$ (resp., $p^-$) if, and only if: (i) $G$ contains an edge $(v, v')$ (resp., $(v', v)$) labeled by $p$; (ii) the transition function $\delta$ contains the triple $(q, [v'], q')$, and (iii) the node $v$ has been visited at depth $d$. If $\delta$ does not contain the triple $(q, [v'], q')$ then the edge from $(v, q, d)$ to $(v', q, d+1)$ labeled with $p \in \mathcal{L}_e$ (resp., $p^-$) is added to $G \times A_t$. The selection of the edges of $G$ to be traversed and the nodes/edges to be added to $G \times A_t$ is made at lines 6-11. The function `expandState` (see Algorithm 2) (lines 3,6) drives the traversal of the data graph according to the transitions of the automaton $A_t$.

Note that an early termination condition is implemented in Algorithm 1 line 7 by: *(i)* limiting the horizon of the traversal to $h$ and *(ii)* stopping the traversal in advance as soon as some node is reached at a depth that does not allow to reach all the remaining entities of the input tuple. Indeed when the state $q_j$ is reached, it is necessary to perform at least $m$-1-$j$ additional traversals to reach the final state, and thus the entity $e_m$. It is easy to see that the size of $G \times A_t$ is linear both in the size of $G$, the size of the tuple-automaton and the horizon h, i.e., $|G \times A_t| = \mathcal{O}(|G| \times |A_t| \times h) = \mathcal{O}(|G| \times m \times h)$.

**Lemma 5** There exists a path of length at most $h$ connecting $e_1$ to $e_m$ in $G$ and passing, in order, through $e_2, ..., e_{m-1}$, if, and only if, there exists a path from $(e_1, q_0, 0)$ to $(e_m, q_{m-1}, l)$ in the graph $G \times A_t$ such that $l \leq h$.

By leveraging the above property, Algorithm 3 uses $G \times A_t$ to build the $m$-meta structure instance $s$. The idea is to start with an empty $m$-meta structure instance and navigate $G \times A_t$ backward (from $(e_m, q_{m-1}, l)$ to $(e_1, q_0, 0)$) by adding nodes and edges to $s$ (lines 5, 7 and 8). Each node and each edge of $G \times A_t$ (in the opposite direction) is visited at most once with cost $\mathcal{O}(|G \times A_t|) = O(|G| \times |A_t| \times h) = O(m \times h \times |G|)$. Thus, the total cost, when also considering the cost of building $G \times A_t$, is $O(m \times h \times |G|)$

The above algorithm uses a horizon $h$ to only consider paths of length at most $h$ interlinking the entities in $t$. We now discuss a variant, which introduces a generic top-$k$ path filtering mechanism based on an edge weighting function.

Edge weighting function. To filter the $m$-meta structure instance found by the base algorithm described above we use an approach that assigns to each edge label a weight. Weights can be assigned according to several strategies; in this paper we use informativeness, and specifically we build upon the notion of Inverse Triple Frequency (`ITF`) introduced and evaluated in our previous work [16]. Basically, the less frequent an edge label is the more it is informative thus getting a higher weight. More formally, for an edge label $p$ in $G$ we have that $\texttt{ITF}(p, G) = log\ \frac{|\mathcal{E}|}{|\mathcal{E}|_{\pi(p)}}$, where $|\mathcal{E}|_{\pi(p)}$ is the number of statements in $G$ where $p$ appears. Note that `ITF` values can be precomputed offline. Since our top-k algorithm works by extracting minimum cost paths, we assign lower `ITF` values to more informative edge labels.

**Input** : $G \times A_t$: Marking of $G$ with states of $A_t$, integer $k$
**Output**: $s = (N_s, M_s) \subseteq G$: $m$-meta structure Instance
1: $H = []$ /* Heap used to store prioritized paths */
2: $N_s$=$M_s$=$\emptyset$
3: **for all** $(v, q, i)$ in $G \times A_t$ **do**
4: $count_{(v,q,i)}$=0 /* number of times a node $(v, q, i)$ in $G \times A_t$ is visited */
5: $P_{(e_1,q_0,i)} = \{(e_1, q_0, i)\}$
6: $H$.add($P_{(e_1,q_0,i)}$,0) /* the initial total weight of the path is 0 */
7: **while** $H \neq \emptyset$ and $\sum_i count_{(e_m,q_{m-1},i)} < k$ **do**
8: $(P_{(v,q,i)}$,C)=$H$.`extractMinCost`() /* extract the path with minimum cost C */
9: $count_{(v,q,i)} = count_{(v,q,i)} + 1$
10: **if** $v = e_m$ and $q = q_{m-1}$ **then**
11: $s$.`add`($P_{(v,q,i)}$)
12: **else if** $count_{(u,q,i)} \leq k$ **then**
13: **for all** $((v, q, i), p, (v', q', j))$ in $G \times A_t$ **do**
14: **if** $(v', q', j) \notin P_{(v,q,i)}$ **then**
15: $P_{(v',q',j)}$=`concatenatePath`($P_{((v,q,i))}((v, q, i), p, (v', q', j))$)
16: $H$.add($P_{(v',q',j)}$,$C + \mathbf{ITF}(p, G)$ /* insert the path */

**Algorithm 4:** `selectTopK`($G \times A_t$, $k$)

Edge Weight Based Algorithm. We describe a variant of the base algorithm that exploits edge label informativeness. After building the $m$-meta structure instance via the base algorithm, `selectTopK` (Algorithm 4) is used to build the top-$k$ $m$-meta structure instance as the graph obtained by considering the top-$k$ *most informative paths* between $(e_1, q_0, 0)$ and $(e_m, q_{m-1}, l)$ (line 11). Algorithm 4 is an adaptation of Eppstein's algorithm [3] to find $k$ shortest paths in a graph, where each node can be visited at most $k$ times.

**Lemma 6** Algorithm 4 runs in $\mathcal{O}(|edges(G \times A_t)| + k \times |nodes(G \times A_t)| \times \log |nodes(G \times A_t)|) = \mathcal{O}(k \times m \times h \times |G| \times \log(m \times h \times |G|))$ where $nodes(G)$ is the set of nodes in $G$ and $edges(G)$ the set of edges.

### 2.3 SP2: Abstracting Meta Structure Instances

The second step to solve the $m$-MetaStructureComputation is the abstraction of an instance $s$ into an $m$-meta structure $\mathcal{S}$ by using the KG schema $T_G$. We considered typing information (the `type`) of the nodes in the $m$-meta structure instance. Existing methods (e.g., [12, 21]) often assume that each node in a KG $G$ belongs to exactly one class; hence, to abstract $s$ it is enough to substitute to each node in $s$ its class. However, in complex and real KGs, nodes can belong to multiple classes. Hence, our approach assign to each node in $s$ the Lowest Common Ancestor (LCA) of all its types in the type hierarchy. We also considered another strategy based on the `domain` and `range` of edge labels. Given a node $n \in s$, we consider all its incoming and outgoing edges; then, by considering their `range` and `domain`, we obtain a set of types. The type of $n$ in the meta structure $S$ will be the LCA of the types in this set.

## 3 Meta Structure Based Relevance

This section outlines an $m$-meta structure-based relevance measure called Layered Tuple Relevance (LTR). Given a KG $G{=}(V, E)$ and an $m-$meta structure $\mathcal{S}$ including $Q$ nodes, the relevance between a tuple including *at most* $Q$-1 distinct entities and a target entity $e_Q$ is defined as follow:

$$\mathcal{R}[(e_1, e_2, ..e_{Q-1}), e_Q \mid \mathcal{S}] = \sum_{s \text{ instance of } \mathcal{S}} f[(e_1, e_2, ..e_{Q-1}), e_Q \mid s]$$

where $f$ is a relevance measure and $s$ is an instance of the $m$-meta structure $\mathcal{S}$ (see Definition 3). One basic form of $\mathcal{R}$ would be to simply *count* the number of instances that the tuple $(e_1, e_2, ..e_{Q-1})$ and the target entity $e_Q$ share; $f$ would simply return 1 for each instance $s$ of $\mathcal{S}$ matching the tuple $(e_1, e_2, ..e_Q)$. For instance, $\mathcal{R}[(\text{A. Aho}, \text{J. Hopcroft}), \text{D.Knuth} \mid \mathcal{S}]$ where $\mathcal{S}$ is shown in Fig. 1 (a) gives 11 instances; apart from the Turing Award, shown in the instance in Fig. 1 (b), there are 10 more awards, among which the Faraday Medal and Kyoto Prize. Other entities that are relevant to the input tuple (A. Aho, J. Hopcroft) are I. Sutherland, J. Ullman and T. Hoare, for which there are 16, 12, and 9 instances, respectively. Using count leads to biased results for two main reasons: (i) count is not bound, so it is difficult to have an objective way of interpreting relevance; (ii) count favors popular objects, as objects with large degrees lead to a larger number of instances. LTR is bound between 0 and 1 and takes into account the specificity of $m$-meta structure instances in the relevance assessment.

LTR splits a $m$-meta structure $\mathcal{S}$ in two parts $\mathcal{S}_l$ and $\mathcal{S}_r$. $\mathcal{S}_l$ considers the subgraph of $\mathcal{S}$ obtained by removing the node where the target entity $e_Q$ is mapped and its edges; besides, in $\mathcal{S}_l$ the $Q$-1 entities in input are used in lieu of their types. $\mathcal{S}_r$ only retains the node (i.e., the type) where $e_Q$ is mapped and its immediate neighbors. Splitting $\mathcal{S}$ in this way models the fact that we are interested in the relevance between at most $Q$-1 entities whose structural information is captured by $\mathcal{S}_l$ and a target entity $e_Q$ whose structural information is captured by $\mathcal{S}_r$. We sketch the rationale behind LTR via an example.

Consider the $m$- meta structure $\mathcal{S}_t$ in Fig. 1 (a) including $Q{=}5$ nodes. We are interested in measuring the relevance between the pair A. Aho (aa), J. Hopcroft (jh) and an instance et of Scientist (i.e., $\mathcal{S}_3$ in Fig. 4), that is, $\mathcal{R}_{\text{LTR}}[(\text{aa}, \text{jh}), \text{et} \mid \mathcal{S}_t] = \sum_s \text{LTR}[(\text{aa}, \text{jh}), \text{et} \mid s]$ with $s$ being an instance of $\mathcal{S}_t$. Here, we are instantiating 2 out of the 4 possible nodes in the $m$-meta structure. Although in this example we focus on aa and jh we may use *any pair of entities* (or tuple of at most 4 entities) that conform to the $m$-meta structure in Fig. 1 (a). Fig. 4 (a) shows $\mathcal{S}_l$ (dotted blue line) and $\mathcal{S}_r$ (dotted orange line) while Fig. 4 (b) shows instances of both Award and Scientist; entities for the relevance assessment are instances of the node $S_3$. LTR leverages the tree structure shown in Fig. 4 (c). The root (level 0) is $\mathcal{S}_l$ and is used to start the traversal of the KG by creating at level 1, for each possible pair of instances of $A_1$ and $A_2$, a new child node; anecdotally there are 10 children in this example; IEEE J. von Neumann Medal and Turing Award, are examples of instances of $A_1$ and $A_2$, respectively. Each child node gives a new instance of the (sub)-meta structure $\mathcal{S}_l$. For each mapping $a_1$ and $a_2$ of $A_1$ and

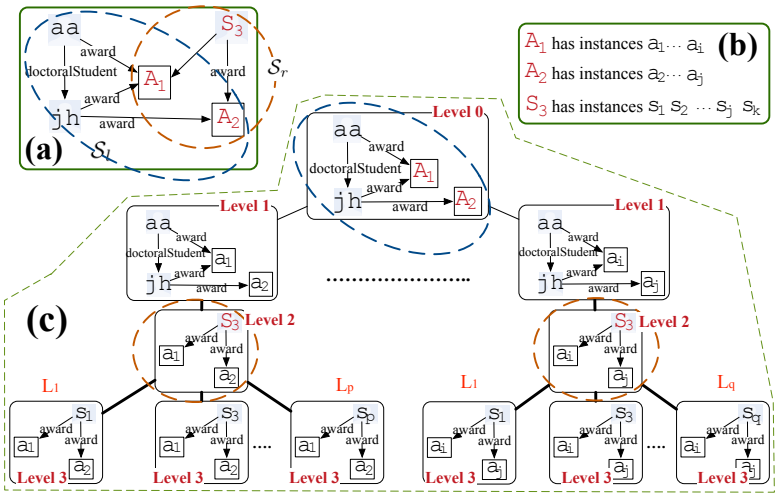


**Fig. 4.** Computing meta structure-based relevance via LTR.

$A_2$, a new child node is created at level 2 by instantiating these mappings into $\mathcal{S}_r$. Leaves (level 3) are created by instantiating into $S_3$ the instances $s_1$, ...,$s_k$ obtained again by traversing the KG (level 2). As an example, for the leftmost node at level 2, $S_3$ has $p$ instances and thus the corresponding subtree has $p$ leaves. Note that level 1 and level 3 contain data triples only, while level 0 and level 2 have the placeholders $A_1$, $A_2$ and $S_3$, respectively. These can be basically treated as variables with a typing constraint (e.g., $S_3$ is a Scientist). The relevance between the pair (aa, jh) and a target entity et is computed starting from the leaves of the tree (level 3) and checking for each leaf whether et appears. As an example, if et=$s_3$, the second leftmost leaf and the central leaf from the right hand side subtree receive a value 1; all the others receive 0. Relevance is assessed via equation 1, where $N_i$ is the number of nodes at level $i$, $N_n$ is the number of nodes in the subtree rooted at $n$ (excluding $n$), and $N_{n|1}$ is the number of nodes in the subtree rooted at $n$ having value 1 (excluding $n$).

$$R_{\texttt{LTR}} = \frac{\sum_{n \in level2} \Theta(n)}{N_1} \quad \text{with} \quad \Theta(n) = \frac{N_{n|1}}{N_n} \tag{1}$$

When traveling up the tree starting from the leaves, each node $n$ at level 2, receives the value $N_{n|1}/N_n$. Relevance is computed at the root by summing values of nodes at level 2 and dividing by the number of children at level 1. This guarantees that target entities sharing with the other entities of the tuple less frequent objects (i.e., an Award $a_i$) are ranked higher. Hence, we have that differently from the count based relevance measure, T. Hoare is ranked higher than J. Ullman because he received the T. Award, which is a less common than the ACM Fellowship shared with J. Ullman. In Section 4.2 we will show the usefulness of LTR in a variety of knowledge discovery tasks.

## 4 Implementation and Evaluation

We implemented the algorithms in Java and the interface of MEKoNG in Java FX. The algorithm to compute and abstract $m$-meta structures discussed in Section 2 works in main memory whereas the `LTR` measure has been implemented by using a combination of Java code and SPARQL queries. We considered different KGs in our experiments; (i) **DBLP**: the dataset described in Huang et al. [10], which contains ∼50K nodes and ∼100K edges and includes four types of entities (Paper, Author, Venue, Topic); (ii) **YAGO**: Yago core, which consists of 5M edges, 125 types and ∼2M nodes having ∼365K types; (iii) **DBpedia**: a subset including ∼2M nodes and ∼5M edges obtained from classes such as Person, Location, and City. For the experiments about relevance (Section 4.2) the full datasets have been accessed via their SPARQL endpoints. Experiments have been performed on a MacBook Pro with a 2.8 GHz i5 CPU and 16GBs RAM. Results are the average of 5 runs. We abstract $m$-meta structure instances using the LCA of the types of each entity.

### 4.1 Efficiency

To test efficiency we used increasing subsets of Yago (from 1.5M of edges to the full dataset) and the full DBLP. Entities are randomly chosen for each run. Fig. 5 (a) reports the average running time when varying the number of input entities $n_e$ w.r.t. the depth spanning from $n_e$-1 to 6.

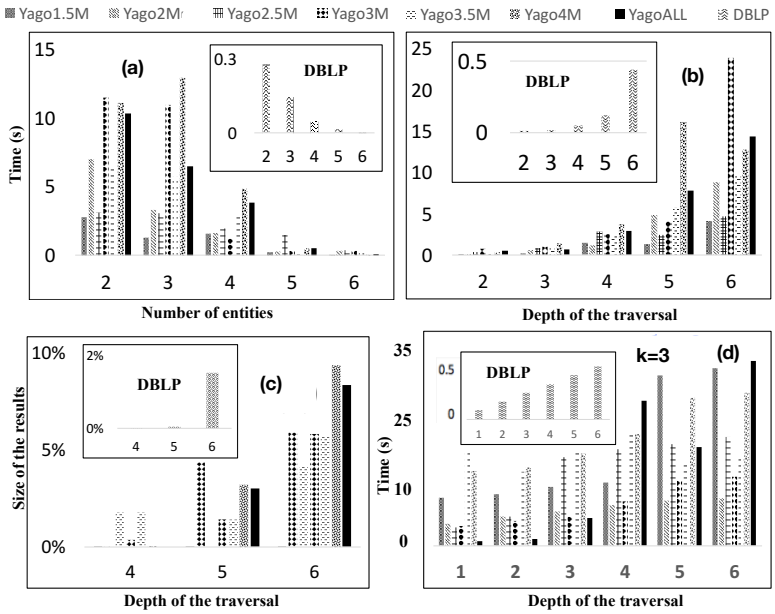


**Fig. 5.** Results about efficiency.

We observe that in general the running time does not strictly depend on the size of the dataset; it actually depends from the size of the input tuple with lower values being responsible for higher running times. The reason is that the early stopping condition in Algorithm 1 (line 7) is satisfied more frequently when the number of input entities increases (for a given depth). Fig. 5 (b) reports the average running times when varying the depth $d$ of the traversal w.r.t. the number of entities spanning from 2 to $d$+1. The running time increases with the depth (as one would expect); it reaches its maximum value for the subset of Yago including 3M of edges. Fig. 5 (c) reports the size of results as a function of the depth of the traversal. The size is measured as the % of the triples in the whole dataset that belong to the $m$-meta structure discovered. We note that the higher the depth the larger the $m$-meta structure discovered. Nevertheless, the size always remains below 10%. Results for depth equals to 2 and 3 are not reported as they approach zero.

As for the algorithm that considers top-k paths only, Fig. 5 (d) reports the average running times as a function of the depth $d$ of the traversal for a fixed $k$ w.r.t. the number of entities spanning from 2 to $d$+1. Running times are higher than those in Fig. 5 (b) since the algorithm requires a further step to find the top-k most informative paths. In particular, for depth equals to 6 the running time is ∼35sec (it was 25sec. without the application of the top-k algorithm). Using the top-k algorithm allows to obtain significantly smaller and more understandable $m$-meta structures As an example, for 3 entities and depth 6, the $m$-meta structure instance on DBLP has 390 nodes and 928 edges when using the base algorithm; the number of nodes is 10 and the number of edges is 11 when using the top-$k$ algorithm instead. On the whole Yago dataset, for the same depth and number of entities, the number of nodes and edges is ∼8K and ∼15K, respectively, for the base algorithm; these numbers become 8 and 10 for the top-k. Running times for DBpedia are not reported since they showed similar trend to that of Yago. Overall, the base algorithm is able to retrieve an $m$-meta structure instance linking the input tuple in a reasonable amount of time (considering the size of the dataset and the depth); however, the size of such $m$-meta structure instance can become prohibitively large. On the other hand, the top-k requires a slightly larger running time (with an increase of about 30% on average) but allows to obtain smaller and more useful $m$-meta structure instances.

### 4.2 Effectiveness

We now compare the performance of `LTR` with the approach that count meta structure instances (referred to as `StrCnt`) and `SCSE` and `BSCSE` [10]; `SCSE` performs subgraph expansion (from a source entity) by restricting the traversal of a KG to $\mathcal{S}$) and `BSCSE` combines and generalizes `StrCnt` and `SCSE`. In this case we consider tuples consisting of two entities. Fig. 6 reports some of the meta structures used to test the effectiveness of the `LTR` measure. Besides meta structures we also considered (combinations of) some of their constituent meta paths.

**Entity Resolution.** We used `LTR` to identify entities in a KG that refer to the same person. In order to construct the ground-truth, we used the entity tuple

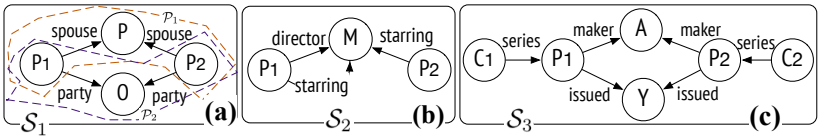


**Fig. 6.** Meta structures used in the experiments.

(Barack Obama, Republican Party, Presidency of Barack Obama) to obtain the meta structure in Fig. 6 (a) using data from DBpedia. The meta structure tells us that two entities of type Person (i.e., $P_1$ and $P_2$) are both married to the same person P and member of the same Organization (i.e., O); Fig. 6 (a) also shows (dotted lines) two meta paths $\mathcal{P}_1$ and $\mathcal{P}_2$.

By using this meta structure we obtained a set of 558 entity pairs; then, we manually inspected these results to discover 124 pairs of entities that refer to the same person (e.g., Barack Obama, Presidency of Barack Obama). As for Yago, we used the ground-truth constructed by Huang et al. [10] including 44 positive pairs and 2967 negative ones. The meta structure used in Yago is slightly different; it uses marriedTo in lieu of spouse and affiliatedTo in lieu of party. We compared the performance of `LTR`, `StrCnt`, `SCSE`, `BSCSE` on the two meta paths $\mathcal{P}_1$ and $\mathcal{P}_2$ taken separately and their (optimal) linear combination. Results are shown in Table 1. Using meta paths alone gives the lowest performance, while their combination brings slightly better results on DBpedia. The reason is that meta paths fail in capturing complex relationships that meta structures can capture using shared nodes. `StrCnt` gave better results but it favors popular objects thus giving higher relevance to pairs of entities sharing more meta structure instances. `LTR` performs better as it is able to perform a deeper assessment of relevance by considering the specificity of entities shared (e.g., P and O) in a meta structure. `SCSE`/`BSCSE` work on main memory and could not handle DBpedia because of a memory overflow. On Yago, the trend of results remains the same as DBpedia with `LTR` reporting slightly lower performance than `SCSE`/`BSCSE`. We can explain this behavior by the fact that `SCSE`/`BSCSE` perform a finer-grained layered structural analysis of a meta structure; however, they assume that the meta structure itself is a DAG. `LTR` does not impose this constraint and performs a higher level analysis by splitting a meta structure in two parts (see Section 3). Nevertheless, `LTR` brings the following advantages over `SCSE`/`BSCSE`: (i) it can work on any existing KG exposed via SPARQL, while `SCSE`/`BSCSE` require preprocessing and index building/maintenance; (ii) it can work with arbitrarily-shaped meta structures and not only DAGs; (iii) `LTR` is built given a *tuple of entities* while `SCSE`/`BSCSE` assume meta structures are given.

**Table 1.** AUC for the Entity Resolution Experiment.

| Dataset | $\mathcal{P}_1$ | $\mathcal{P}_2$ | $\alpha\mathcal{P}_1+(1-\alpha)\mathcal{P}_2$ | `StrCnt` | `SCSE` | `BSCSE` | `LTR` |
|---|---|---|---|---|---|---|---|
| Yago | 0.223 | 0.115 | 0.298 | 0.495 | 0.534 | 0.543 | 0.512 |
| DBpedia | 0.167 | 0.118 | 0.213 | 0.314 | – | – | 0.498 |

**Table 2.** Top-5 relevant entities using different meta structures and source entities.

| | Meta Structure | | | |
|---|---|---|---|---|
| | $\mathcal{S}_2$ (Fig. 6 (b)) | $\mathcal{S}_2$ (Fig. 6 (b)) | $\mathcal{S}_3$ (Fig. 6 (c)) | $\mathcal{S}_3$ (Fig. 6 (c)) |
| **Rank** | $P_1$=Q. Tarantino | $P_1$=C. Eastwood | $C_1$= ISWC, Y=2010 | $C_1$= ISWC, Y=2016 |
| **1** | H. Keitel | S. Locke | ESWC | ESWC |
| **2** | S. L. Jackson | K. Costner | EKAW | WWW |
| **3** | M. Madsen | M. Freeman | WWW | JIST |
| **4** | T. Roth | M. Hill | Description Logics | EKAW |
| **5** | U. Thurman | J. Walter | I-Semantics | swat4ls |

**Ranking.** We now discuss relevance in different domains. The meta structure in Fig. 6 (b) models relevance with a person ($P_1$) that has acted and directed movies (M) where also acted a different persons ($P_2$). The meta structure in Fig. 6 (c) is used to asses relevance on DBLP based on the fact that authors (A) have published two papers ($P_1$ and $P_2$) in two venues ($C_1$, $C_2$) in the same year (Y). Table 2 reports the top-5 relevant entities for different source entities (i.e., instantiations of nodes in a meta structure).

As for $\mathcal{S}_2$, in DBpedia Q. Tarantino is highly related to H. Keitel and S. L. Jackson. On Yago (full ranking not reported) we have that S. L. Jackson is ranked higher than H. Keitel and that L. Bender enters the top-5. By changing the source entities to C. Eastwood, on DBpedia we get S. Locke and then K. Costner while in Yago (ranking not reported) we have M. Freeman ranked second and A. Her entering the ranking. $\mathcal{S}_3$ uses two entities as source (a venue $C_1$ and a year Y) and retrieve the top-k most related venues (instances of $C_2$). In 2010, ESWC is the most relevant and I-Semantics is the least relevant according to the meta structure $\mathcal{S}_3$. Interestingly, when changing year we can see that WWW becomes more relevant and that JIST in included in the top-5. `LTR` offers a flexible way of assessing relevance by allowing to fix a subset of entities in a tuple. Fixing two entities allowed to obtain a more refined (venue-year-centric) ranking than fixing only the venue. In this latter case the ranking would have been different: OWLED would have entered the ranking in lieu of swat4ls. As an example, by using `StrCnt` in 2016 we would have obtained BigData instead of swat4ls, although the latter (Semantic Web Applications and Tools for Life Science) is clearly more relevant. Overall, `LTR` coupled with meta structures offers flexibility in two respects: (i) it can be applied in a variety of KGs thanks to its SPARQL-based implementation; (ii) an arbitrary subset of nodes in a meta structure (e.g., a venue and a year) can be chosen as a source for relevance wrt a target node (e.g., another venue).

## 5 Related Work

**Connectivity Structures.** The problem of finding connectivity structures in graphs has been studied in different fields [19]. Hintsanen [9] focused on finding the most reliable subgraph in a graph subject to edge and node failures. Ramakrishnan et al. [18] focused on finding the most informative entity-relationship sub-

graphs in a given graph. A variant of this problem has been studied by Mendes et al. [14]. Other approaches have focused on determining specific substructures such as the minimum spanning tree or approximations (e.g., STAR [11]). Neither the above approaches focus on finding meta structures (schema graphs) given a tuple of entities nor on using meta structures for relevance computation. A number of approaches (e.g., [2, 4, 6, 8, 17]) have reduced the problem of finding paths (and meta paths) between entities to that of directly querying a KG by fixing a maximum path length and then displaying/abstracting (a subset of) the paths found. Beside the fact that these approaches neither focus on meta structure nor on relevance computation, their major drawback is that they require to enumerate paths first. A few approaches focused on finding meta paths; Lao et al [12] tackle this problem by using constrained random walks. AMIE [7] is a system to mine association rules from KG. The difference with meta paths algorithms, and with our meta structure finding algorithm, is that AMIE does not find associations by taking into account the user input (i.e., a tuple of entities). Meng et al. [15], focuses on meta paths while we focus on meta structures. A recent approach [1] focuses on finding associations given a tuple of entities. Our work differs in several respects: (i) we control the size of the subgraph linking the input tuple by including top-k informative paths since this subgraph can be very large and thus difficult to visualize (e.g., for relatedness explanation) and reuse (e.g., for relevance computation); (ii) we do not focus on trees as meta structures are graphs; (iii) we introduce a novel meta structure-based relevance measure and show its usefulness in a variety of tasks. Overall, we tackle a more comprehensive problem: *finding meta structures given a tuple of entities as input*, *meta structure-based relevance*, and *user supports* (via MEKoNG).

**Relevance Measures.** Several measures have been proposed to compute relevance; examples include: (i) measure based on the graph structure (e.g., common neighbors), Jaccards coefficient or based on random walks [12]; (ii) schema-based measures based on meta-paths (e.g., [20]); (iii) Huang et al. [10] define relevance based on meta structures. Our work differs from (i) in the fact that `LTR` leverages schema information and from (ii) because we use meta structures instead of meta paths. As for (iii), the underlying assumption that meta structures are already available may be not realistic for a number of reasons: manually defining meta structures can be tedious and difficult when dealing with complex KGs like Yago/DBpedia; and complex meta structures can be difficult to discover, especially if this is done without automation. Our work is more comprehensive as it deals with both meta structure finding and relevance computation. Finally, differently from (i), (ii), and (iii) we focus on entity tuples and not just pairs.

## 6 Concluding Remarks and Future Work

We discussed an approach to compute meta structures combining an automata-based algorithm and its variant, which considers the most informative top-k paths. As our algorithm to find meta structures works in main memory, it cannot deal with very large KGs. To address this limitation we plan to consider the

vertex-centric Gather Apply Scatter (GAS) paradigm [13] in the future. We have shown how meta structure-based relevance is useful in a variety of task (e.g., entity resolution, ranking). Our implementation of `LTR`, by a combination of Java code and SPARQL queries, makes it readily available on any SPARQL endpoint. Testing `LTR` in other domains is in our research agenda.

## References


1. G. Cheng, D. Liu, and Y. Qu. Efficient Algorithms for Association Finding and Frequent Association Pattern Mining. In *ISWC*, 119–134, 2016.
2. G. Cheng, Y. Zhang, and Y. Qu. Explass: Exploring Associations between Entities via Top-K Ontological Patterns and Facets. In *ISWC*, 422–437. 2014.
3. D. Eppstein. Finding the k Shortest Paths. *SIAM J. Comp.*, 28(2):652–673, 1998.
4. V. Fionda, G. Pirrò, and C. Gutierrez. Building Knowledge Maps of Web Graphs. *Artificial Intelligence*, pp. 143-167, 2016.
5. V. Fionda, G. Pirrò, and C. Gutierrez. NautiLOD: A Formal Language for the Web of Data Graph. *ACM Trans. on the Web*, 9 (1), 2015.
6. V. Fionda, G. Pirrò, M. Consens. Extended Property Paths: Writing More SPARQL Queries in a Succinct Way. In *AAAI*, 2015.
7. L. A. Galárraga, C. Teflioudi, K. Hose, and F. Suchanek. AMIE: Association Rule Mining Under Incomplete Evidence in Ontological Knowledge Bases. In *WWW*, 413–422, 2013.
8. P. Heim, S. Hellmann, J. Lehmann, S. Lohmann, and T. Stegemann. RelFinder: Revealing Relationships in RDF Knowledge Bases. In *S. Multimedia*, 182–187, 2009.
9. P. Hintsanen. The Most Reliable Subgraph Problem. In *PKDD*, 471–478, 2007.
10. Z. Huang, Y. Zheng, R. Cheng, Y. Sun, N. Mamoulis, and X. Li. Meta Structure: Computing Relevance in Large Heterogeneous Information Networks. In *KDD*, 1595–1604, 2016.
11. G. Kasneci, M. Ramanath, M. Sozio, F. M. Suchanek, and G. Weikum. STAR: Steiner-Tree Approximation in Relationship Graphs. In *ICDE*, 868–879, 2009.
12. N. Lao and W. W. Cohen. Relational Retrieval Using a Combination of Path-Constrained Random Walks. *Machine Learning*, 81(1):53–67, 2010.
13. M. Grzegorz. and M. Austern et al. Pregel: a System for Large-Scale Graph Processing. *SIGMOD*, 135–146, 2010.
14. P. N. Mendes, P. Kapanipathi, D. Cameron, and A. P. Sheth. Dynamic Associative Relationships on the Linked Open Data Web. In *Web Science Conference*, 2010.
15. C. Meng, R. Cheng, S. Maniu, P. Senellart, and W. Zhang. Discovering Meta-Paths in Large Heterogeneous Information Networks. In *WWW*, 754–764, 2015.
16. G. Pirrò. REWOrD: Semantic Relatedness in the Web of Data. In *AAAI*, 2012.
17. G. Pirrò. Explaining and Suggesting Relatedness in Knowledge Graphs. In *ISWC*, 622–639, 2015.
18. C. Ramakrishnan, W. H. Milnor, M. Perry, and A. P. Sheth. Discovering Informative Connection Subgraphs in Multi-Relational Graphs. *SIGKDD Newsletter*, 7(2):56–63, 2005.
19. A. Sheth, et al. Semantic Association Identification and Knowledge Discovery for National Security Applications. *JDBM* , 16(1):33–53, 2005.
20. C. Shi, X. Kong, Y. Huang, P. S. Yu, and B. Wu. HeteSim: A General Framework for Relevance Measure in Heterogeneous Networks. *TKDE*, 26(10):2479–2492, 2014.
21. Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. PathSim: Meta Path-based top-k Similarity Search in Heterogeneous Information Networks. *PVLDB*, 2011.