WiSP: Weighted Shortest Paths for RDF Graphs

Gonzalo Tartari and Aidan Hogan

IMFD Chile & Department of Computer Science, University of Chile

Abstract. An important aspect of exploratory search over graph data is to understand what paths connect a given pair of nodes. Since the resulting paths can be manifold, various works propose ranking paths likely to be of interest to a user; these methods often rely on enumerating all such paths (up to a fixed length or number) before ranking is applied. In this paper, we instead propose applying a shortest path search on weighted versions of the graph in order to directly compute the most relevant path(s) between two nodes without fixed-length bounds, further obviating the need to enumerate irrelevant paths. We investigate weightings based on node degree, PageRank and edge frequency, contrasting the paths produced by these schemes over the Wikidata graph and discussing performance issues. Finally we conduct a user study over Wikidata where evaluators assess the quality of the paths produced; though inter-rater consensus on which paths are of most interest is low, we achieve statistically significant results to suggest that users find the weighted shortest paths more interesting than the baseline shortest paths without weights.

1 Introduction

Though a wealth of data has been published as Linked Data using the Semantic Web standards, we still lack techniques and tools by which non-expert users can interact with graph-structured data [20]. A characteristic feature of interacting with such graphs is the ability to query the connectivity of nodes through paths of arbitrary length. Along these lines, a number of proposals have been made to empower users to find such paths satisfying certain conditions, such as shortest paths, or simple paths that do not revisit a node (or edge), or to find pairs of nodes connected by paths satisfying regular expressions [2]. However, in RDF graphs, many shortest paths will pass through nodes of high-degree (e.g., a country, a type) and thus may be of little interest; on the other hand, in exploratory search scenarios, users may not have specific path expressions in mind, but may simply wish to find noteworthy paths between two nodes of interest.

Hence further work has been conducted on ranking *simple paths*[1] – paths not repeating a node – that exist between the query nodes according to a variety of metrics and methods [1,3–5,7,8,19,23,27–29]. A key limitation of such approaches is that they often require enumerating all simple paths between the query nodes prior to ranking. Given that there is a potentially factorial number of simple

[1] Aka. s–t paths, semantic associations, complex relations, etc.

paths[2] – where even counting such paths between two nodes is known to be #P-complete [26] – a pragmatic threshold must be applied to either the number of paths or the length of paths considered; even with these thresholds, a large number of irrelevant paths may still need to be considered.

In this paper, we thus propose to reformulate the problem of ranking simple paths to that of computing shortest paths in a weighted RDF graph;[3] our core hypothesis is that *we can design a weighting scheme for an RDF graph such that the shortest paths in the weighted version of the graph are of more interest to the user than in the non-weighted version* (note that the weights only optionally form part of the output: the hypothesis considers "plain" paths being output).

The benefit of such a weighting scheme can be seen from two perspectives. First, with respect to ranking paths, simple paths of relevance to the user can be directly computed using well-known shortest path algorithms without having to explicitly enumerate a potentially exponential number of paths, and without *a priori* bounds on the length or number of paths considered. Second, with respect to shortest paths, a potentially factorial number of shortest paths can still exist between two nodes, where standard shortest path algorithms either return the first such path found, or return all shortest paths; a weighting scheme as we propose could thus be used to "break ties", providing a more granular notion of (weighted) shortest path than considering path length alone.

In Figure 1, we provide a motivating example taken from the Wikidata knowledge graph [30]; note that for readability we present the example with node identifiers based on English labels (e.g., `:film`, `:director`) rather than Wikidata identifiers (e.g., `:Q11424`, `:P57`). Let us consider a user that wishes to find connections between the two theatrical movies `:Blade_Runner` and `:Top_Gun`. First we observe that paths should traverse edges in either direction; for example, we consider both `:director` and `:directed`$^-$, returning the direction traversed in the output. Second, we consider simple paths that do not visit the same node twice since, otherwise, we would have potentially infinite paths with redundancy (e.g., jumping back and forth between the Scott brothers an arbitrary number of times). Thereafter, we can identify six simple paths: one stating that both movies are instance of `:film`, one stating that their country of origin is the `:United_States_of_America`, and four indicating that they were directed by brothers (two paths for each direction of the `:sibling` relation and its inverse). Though the paths between the query nodes through `:United_States_of_America` or `:film` have the shortest length, we argue that the longer path through the Scott brothers is more likely to be of interest to a user.

While many works propose to first enumerate all paths and thereafter rank them to try find the most interesting ones, we rather propose to apply a weighting to the graph in an off-line phase and then compute the shortest paths in the weighted graph on-line. For example, one could imagine that high weights are

[2] In a k-clique ($k \geq 2$) with 2 directed labelled edges between each node pair, there are $4^l \frac{(k-2)!}{(k-1-l)!}$ simple paths of length l (encoding edge labels and inverses) from any node to any other; for $l = k-1$, this gives $4^{k-1}(k-2)!$ simple paths.

[3] A shortest path in a weighted graph has the lowest sum of node and edge weights.

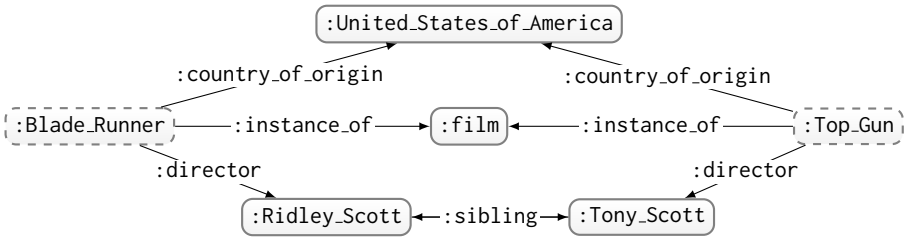


Fig. 1. Sample sub-graph from Wikidata involving two theatrical films; for readability, we use node identifiers based on labels; query nodes are dashed

assigned to high-centrality nodes such as `:film` and to frequent edges such as `:instance_of`; however, care needs to be taken since a naive weighting scheme will lead to long paths through nodes representing relatively obscure entities: the most specific or "improbable" paths are not always best. Furthermore, in other cases, paths of interest may pass through high-centrality nodes, particularly if the edge label is selective (e.g., `:head_of_state`). Hence designing an appropriate weighting scheme requires balancing potentially numerous factors [8].

In this paper, we thus investigate methods by which we can compute simple paths that exist between two input nodes in a directed edge-labelled graph (such as an RDF graph) that are likely to be of relevance to a user. We reduce the ranking problem to that of weighting the nodes and edges of the graph prior to applying a shortest-path algorithm. We evaluate various weighting schemes over the Wikidata graph, comparing the paths that they produce and their performance at various scales, further conducting a user study where evaluators assess the weighted shortest paths given by the various weighting schemes.

Outline: The following section presents related works while Section 3 introduces preliminaries. Section 4 presents our weighting schemes and implementation. Turning to evaluation, Section 5 compares statistics for the paths generated by each scheme over Wikidata, while Section 6 presents the results of a user study. Section 7 concludes and discusses future work.

2 Related Work

Perhaps the most prominent way to traverse paths in RDF graphs is through the property paths feature of the SPARQL query standard, which allows users to find pairs of nodes connected by paths that conform to a specified regular expression; such features are common amongst query languages designed for graphs [2]. The focus of such works is on finding pairs of nodes connected by input-constrained paths rather than finding paths connecting two input nodes.

On the other hand, a number of works have proposed methods to find paths between nodes in RDF graphs. RelFinder [14, 15, 21] finds paths between two nodes indexed by a SPARQL endpoint by issuing a sequence of queries looking for

successively longer paths; while this approach has the benefit of being portable across endpoints, the types of paths that can be found are limited by the use of SPARQL (which cannot return *paths*) and no ranking is applied.

Recognising that the number of paths between two nodes can exceed the user's capacity to comprehend, a wide variety of works have been proposed to rank paths – called *semantic associations*, *semantic relationships*, *complex relationships*, etc. – between nodes. A seminal work is SemRank [3], which proposed a customisable information-theoretic measure to rank paths between nodes possibly restricted to match keywords. Thereafter, a variety of works have been proposed along similar lines, including approaches based on semantic metrics and graph measures [1, 23], inference rules [19], personalisation [28], Bayesian networks [27], ant-colony optimisation [29], informativeness [24], learning-to-rank [4, 5, 7], external rating schemes [16], and so forth; many such methods were recently summarised and evaluated by Cheng et al. [8]. Rather than ranking paths, other works have proposed methods to summarise the paths that exist between two nodes, where RelClus [31] applies hierarchical clustering and Explass [9] applies faceted clusters over paths. All such approaches typically require enumerating the paths between two entities before applying a ranking or clustering step; given that the number of paths is potentially factorial in the number of nodes, these approaches often consider simple paths with a fixed length threshold l or a fixed number of paths k, or some combination thereof.

Other approaches have looked at various problems and applications relating to computing shortest paths for RDF graphs (e.g, [11–13, 17, 22]), but to the best of our knowledge, the only works considering shortest paths over *weighted* RDF graphs are those of Cedeño et al. [6], who deal with application-specific weights such as confidence values;[4] Rusu et al. [25], whose focus is on computing similarity for ontological concepts; and Hulpuş et al. [18], whose focus is on computing the relatedness of nodes for entity disambiguation purposes. Thus, to the best of our knowledge, we are the first work to investigate weighting schemes for RDF graphs such that the resulting weighted shortest paths are of more interest to users than those computed over the unweighted graph.

3 Preliminaries

We now present preliminaries relating to RDF graphs, paths and weights.

RDF graphs: Given pairwise disjoint sets of IRIs ($\mathbf{I}$), literals ($\mathbf{L}$) and blank nodes ($\mathbf{B}$), an *RDF triple* (s, p, o) is an element of $\mathbf{I} \cup \mathbf{B} \times \mathbf{I} \times \mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$, where s is the subject, p predicate and o object. A set of RDF triples G is called an *RDF graph*, owing to the fact that one can view each triple as a directed labelled edge $s \xrightarrow{p} o$, and hence an RDF graph can be viewed as a directed edge-labelled graph [2]. Letting $\pi_{\text{S}}(G) = \{s \mid \exists p, o : (s, p, o) \in G\}$ denote the set of all subjects of G, and letting $\pi_{\text{P}}(G)$ and $\pi_{\text{O}}(G)$ likewise denote respectively the set of all predicates and

[4] Their evaluation considers manually specified weights.

objects of G, we call $\pi_{\text{S}}(G) \cup \pi_{\text{O}}(G)$ the *set of nodes* (aka. *vertices*) of G, and $\pi_{\text{P}}(G)$ the set of *edge labels* (aka. *properties*) of G. Finally, we denote by $G^{\pm}$ the completion of G by inverse edges such that $G^{\pm} := G \cup \{(o, p^-, s) \mid (s, p, o) \in G\}$, where $p^- \notin \pi_{\text{P}}(G)$ is a surrogate edge label used to denote the inverse of p.

Paths: Given an RDF graph G, we call a sequence $v_0 p_1 \dots v_{n-1} p_n v_n$ such that $(v_{i-1}, p_i, v_i) \in G^{\pm}$ for $1 \leq i < n$ a *path* from v_0 to v_n in G; we call n the *length* of the path.[5] We call a path where $v_i \neq v_j$ holds for $1 \leq i < j \leq n$ a *simple path*. We call a path from v to v' a *shortest path* if no other path exists from v to v' with smaller length; all shortest paths are simple. Since we can trivially convert any path from v to v' to a unique path from v' to v by reversing the path and inverting all edge labels, we will often not distinguish the source and target nodes, instead simply referring to the paths *between* two nodes.

Weights: For a graph G, let $V := \pi_{\text{S}}(G) \cup \pi_{\text{O}}(G)$ denote its nodes and $P := \pi_{\text{P}}(G)$ its edge labels. We define a *weighted RDF graph* as a triple (G, ω_V, ω_P), where $\omega_V : V \to \mathbb{R}_{\geq 0}$ and $\omega_P : P \to \mathbb{R}_{\geq 0}$ denote weighting functions on the nodes and edge labels of G respectively; note that for $G^{\pm}$, we will assume ω_P to be defined such that $\omega_P(p) = \omega_P(p^-)$, and that in this current work, we do not consider negative weights. Thereafter, given a weighted RDF graph (G, ω_V, ω_P), we define the weight of a path $v_0 p_1 \dots v_{n-1} p_n v_n$ from G to be the sum of the weights of all intermediate nodes and edge labels: $\Sigma_{i=1}^{n-1} \omega_V(v_i) + \Sigma_{j=1}^{n} \omega_P(p_j)$. A weighted path is a shortest path between two nodes if there does not exist another path between the nodes with lower weight. Note that if we define $\omega_V(v) = 1$ and $\omega_P(p) = 1$ for all $v \in V$ and $p \in P$, shortest paths by weight and length coincide.

Dijkstra's algorithm: To find the shortest paths in a graph, we utilise Dijkstra's algorithm [10]; for reasons of space, here we summarise the algorithm. From a given source node s, the algorithm traverses the graph, updating the distance from s to each node it visits. The closest unvisited node u is always the next to be visited, at which point the distance from u to s ($\delta(s, u)$) is known to be minimal. When node u is visited, the distance of all neighbouring unvisited nodes ($\delta(s, u')$) is updated if the distance is lower than that found previously ($\delta(s, u) + \delta(u, u') < \delta(s, u')$). The algorithm thus visits all nodes once, computing the shortest distance from the source node to all nodes in the graph. When a target node t is provided as input, the search can terminate when the target node is next to be visited; thereafter, the actual path can be constructed by backtracking through the next adjacent node closest to s. Given a weighted RDF graph, we can compute the distance $\delta(v, v')$ between two neighbouring vertices as the sum of the lowest-weighted edge connecting them and the weight of v'. Given a source and target node, efficient implementations of Dijkstra's algorithm using optimal priority queues (to decide the next node to visit) can find a (weighted) shortest path in $O(|G| + |V| \cdot \log|V|)$ time[6] in the worst case.

[5] A sequence $v_0 p v_1$ such that $(v_0, p, v_1) \in G^{\pm}$ is also considered a path of length 1.
[6] By $|S|$ we denote the cardinality of the set S

4 Weighted Shortest Paths

The main goal of this work is to investigate weighting schemes for RDF graphs such that the shortest paths by weight between two nodes are deemed to be of more interest to users than shortest paths by length. Such a scheme would then allow us to compute paths of interest to a user in a best-first manner using shortest path algorithms without having to enumerate all simple paths or having to implement *a priori* bounds on the length or number of paths considered.

As discussed previously, a wide variety of metrics and approaches have been defined for ranking enumerated simple paths that we can adapt for weighting schemes [1, 3–5, 7, 8, 19, 23, 27–29]. These were recently summarised and evaluated by Cheng et al. [8], who mention five data-centric features that may help determine the relevance of a path to a user: *length* (*size*): the length of the path; *frequency*: the number of other nodes that the same edge connects to the same node; *centrality*: where the importance of nodes on the path is considered; *informativeness*: where a path is considered more informative if it is less probable; *specificness*: which, given an ontology, prefers more specific edge-labels (lower in the sub-property hierarchy) to more general ones; and *homogeneity*, which considers whether or not nodes and edges in the path are of the same type.

While our approach will offer various computational benefits and does not require fixed thresholds, since we consider weighted shortest graphs, we are more limited in terms of the ranking functions we can express. More specifically, towards defining such a weighting scheme for an RDF graph G, we are left to instantiate the functions $\omega_V : V \to \mathbb{R}_{\geq 0}$ and $\omega_P : P \to \mathbb{R}_{\geq 0}$ introduced previously for weighting the nodes and edge-labels of the graph. Under our framework, we consider sums of weights on individual nodes/edges, which (currently) rules out adopting measures such as *homogeneity* that consider the diversity of all edges/nodes. Furthermore, we note that there is some redundancy in the above measures, where for example *frequency*, *informativeness* and *specificness* offer different versions of the same underlying principle: how common is a given edge/-path. With these considerations, we propose the following weighting schemes.

*Length (*L*):* We consider as a baseline a weighting scheme where $\omega_V : V \to \{1\}$ and $\omega_P : P \to \{1\}$, setting the weights of all edges and nodes to a fixed constant. This configuration thus produces a shortest path according to length and is equivalent to not considering any weights (on nodes or edges).

*Centrality (*D*,*P*):* In order to weight nodes, we consider two measures of centrality mentioned by Cheng et al. [8]: degree centrality (D), and PageRank (P, aka. Eigenvector centrality). Under degree centrality, the weight of each node is given simply as its (undirected) degree. In order to compute the PageRank centrality of nodes, we consider each triple in the RDF graph G to be a directed edge $s \to o$ and compute PageRank over the resulting graph; thereafter, the weight of each node is given as its PageRank score. Referring back to Figure 1, we would expect nodes such as `:film` and `:United_States_of_America` to have high centrality in a real-world graph with more such triples.

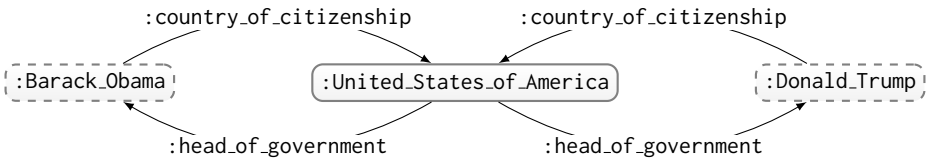


Fig. 2. Sample from Wikidata involving two citizens/presidents of the United States; for readability, we use node identifiers based on labels; query nodes are dashed

*Edge-Label Frequency (*E*):* If we consider Figure 2, where we see four possible paths between the query nodes, we can see that edge labels – and not just nodes – can play a considerable role when we consider which paths are likely to be of interest to a user; in this case, while `:country_of_citizenship` is a relatively common property, `:head_of_government` is, in contrast, relatively rare. Along these lines, we consider an edge weighting based on the frequency of the edge label; more specifically, given an RDF graph G and an edge label $p \in \pi_{\text{P}}(G)$, we define the frequency edge weight as follows: $\omega_P(p) := |\{(s, o) : (s, p, o) \in G\}|$. This measure generally relates to *frequency*, *informativeness*, and *specificness*, though not being directly equivalent to any such measure.

Hybrid: The above schemes may be complementary, dealing respectively with path length, node weights and edge weights. For example, if we consider node centrality alone, the paths produced may tend to be very long, visiting various relatively obscure nodes; furthermore, in Figure 2, we would not be able to distinguish common/uncommon edges to a high-centrality node. Hence we propose a number of hybrid schemes. The first weighting scheme combines the centrality-based node weights and length. We first linearly normalise both ω_V functions to produce scores in the interval $[0, 1]$ such that the maximum value is 1 and the minimum value is 0. Then to combine these weights with length, we propose to sum the respective weights; in other words, we add 1 to the weights of the normalised ω_V such that the interval of values produced is $[1, 2]$, giving rise to two hybrid ranking schemes: DL and PL.[7] While this change appears subtle at first glance, as we will see later in our experiments, this shifting of the range has a major effect: for example, for centrality scores in a $[0, 1]$ range, visiting one high centrality node may have the same cost as visiting thousands of low centrality nodes, whereas when normalised to a $[1, 2]$ range, visiting a high-centrality node has a cost of visiting at most two low-centrality nodes. Finally we likewise normalise ω_E into a $[1, 2]$ interval for the same reasons, producing the final hybrid schemes that combine length, node and edge-weights: DEL and PEL.

User interface: We have implemented a system called WISP: WEIGHTED Shortest Paths. Given an RDF graph, the system creates an in-memory index where nodes

[7] Of course one could consider parameterised hybrid schemes, such as normalising into a $[\alpha, \alpha + 1]$ range ($\alpha > 0$), where as α increases, more weight is put on length. We consider this out of scope but would be interesting to investigate in future.

are identified with integers and point to their neighbours with a labelled edge, allowing graph traversals in both edge directions; node and edge-label weights (computed offline) are stored as an in-memory dictionary. Given two query nodes, Dijsktra's algorithm is then used to find a (weighted) shortest path, returning the first such path found. To facilitate user interaction, we implement auto-complete suggestions that map prefix queries (e.g., `chi`) to node identifiers (e.g., `:Chile/wd:Q298`). The final user-interface – implemented in Javascript (Node.js, React.js and viz.js) – then allows users to search for a source and target node, offering optional features to set the weighting used for the Dijkstra search. The resulting path(s) are then displayed diagrammatically to the user.[8]

5 Statistical Comparison

We now present the results of some experiments of applying our weighted shortest path methods to the Wikidata graph [30]. Wikidata is a collaboratively-edited knowledge base where data relevant to Wikipedia can be curated and managed by human editors. We select Wikidata for our experiments as it is a large graph and – being edited by thousands of users, as well as covering a broad domain of interest – exhibits a high degree of diversity in terms of the types of nodes and edges described, the nature of the relations expressed, and so forth. Thus Wikidata's size poses a non-trivial test of the scalability of our method, while its diversity poses a challenge to the generality of our weightings.

The questions we wish to address with this initial study are as follows:

- How is performance affected by scale and weighting scheme?
- How do the paths produced by different weighting schemes differ?

Data: We perform experiments over the "truthy" version of Wikidata from the 2017-06-07 dump; the truthy version does not include qualifiers or references and selects preferred values for properties (e.g., including only the most recent population for a city). For performance experiments, we define sub-graphs of Wikidata at varying scales: observing that numeric `Q`x IDs are defined in chronological order, where the earliest nodes (often important entities such as countries, prominent persons, etc.) have lower numeric IDs, we set limits on x of 1,600,000, 3,200,000, 6,400,000, 12,800,000. Thereafter, we select all triples (s, p, o) such that both s and o have `Q`x IDs below the limit for x. We apply no limit on property `Q`y IDs and we do not consider edges involving literals or blank nodes. Since IDs are not "dense" in Wikidata, where entities may be removed, applying a limit of 1,600,000 may not result in precisely that number of nodes in the resulting graph. Table 1 presents the number of nodes and edges in each sub-graph used, where the FULL dataset represents the full graph.

Machine: The machine used for all experiments has 2× Intel Xeon Quad Core E5-2609 V3 CPUs (@1.9GHz), 32GB of RAM, and 2× 2TB Seagate 7200 RPM 32MB Cache SATA hard-disks in a RAID-1 configuration.

[8] A demo is available at `http://wisp.dcc.uchile.cl` over a Wikidata sub-graph of 1.2 million nodes; we do not use the full graph for performance reasons discussed later.

Table 1. Wikidata (sub-)graphs used for experiments; FULL indicates the full dataset

Dataset	1.6 M	3.2 M	6.4 M	12.8 M	FULL
Nodes	1,227,382	2,507,582	5,303,322	10,343,129	25,081,334
Edges	6,603,412	12,160,436	22,008,446	36,404,534	89,878,092

Queries: To generate our first set of queries ($\mathcal{Q}_1$), from each dataset, we randomly select 100 pairs of entities. Since pairs selected from larger datasets may not appear in smaller datasets, we also generate a second set common to all graphs ($\mathcal{Q}_2$): we randomly select 100 pairs of entities with ID less than 100,000.

Configurations: We selected L (baseline), D, DL, DEL, P, PL and PEL for testing. Each configuration returns the first shortest path found.

Performance by weights: In Figure 3, we present box-plots of response times for finding a shortest path over the full graph with the selected weighting schemes and $\mathcal{Q}_1$. We can see that the averages and medians are above 27 seconds in all cases, and that the baseline shortest path algorithm is the fastest. The slowest weighting scheme (DEL) is almost twice as slow, taking almost a minute, on average, for each query. Investigating further, for the baseline L scheme, 12.3 million nodes (49% of all nodes) are visited, on average, to answer a query pair; on the other hand, when vertex weights are considered, more nodes are visited, where, for example, P visits 15.2 million nodes (61%), on average.

Performance by scale: The previous experiments suggest that finding shortest paths is a costly process, where a large number of nodes in the graph need to be visited and where the query times may thus exceed a minute in some cases for the full graph. We are thus interested to see how scale affects the performance. In Figure 4, we present the response times for $\mathcal{Q}_2$; for reasons of space, we present results for DEL – the slowest strategy in the previous results – though other strategies followed a similar pattern. We highlight that as the size of each dataset doubles, the mean response times likewise roughly double; an exception is for the FULL dataset, which although more than double the size of the 12.8 M dataset, sees an increase of only $\sim$1.2$\times$ in mean query times, where the average ratio of nodes visited drops from 29% to 23%. Note that we selected $\mathcal{Q}_2$ since we wish to compare different scales for the same set of queries; we also ran $\mathcal{Q}_1$ at all scales for DEL, where the mean times for 6.4 M, 12.8 M and FULL increase to 13.5 s, 34.0 s, and 53.7 s, respectively: nodes randomly sampled from the larger graphs generate longer paths requiring more nodes to be visited.

Comparing weighted shortest paths: Next we look to see if different weighting schemes do in fact generate different paths. In Figure 5, we provide a heat-map that indicates for each pair of weighting schemes the percentage of the 100 $\mathcal{Q}_1$ paths generated over the FULL dataset that coincide with another weighting scheme. In terms of the pairs of schemes producing the most similar results, we

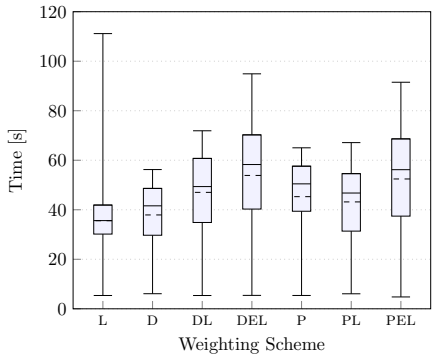


Fig. 3. Response times for $\mathcal{Q}_1$ comparing different weights over the FULL graph

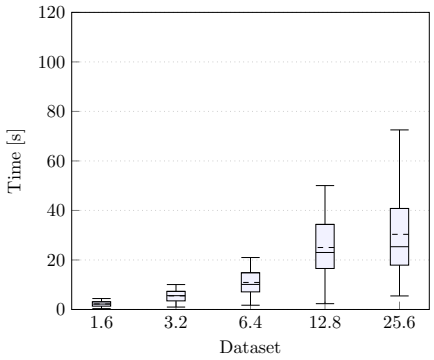


Fig. 4. Response times for $\mathcal{Q}_2$ comparing different scales with DEL weights

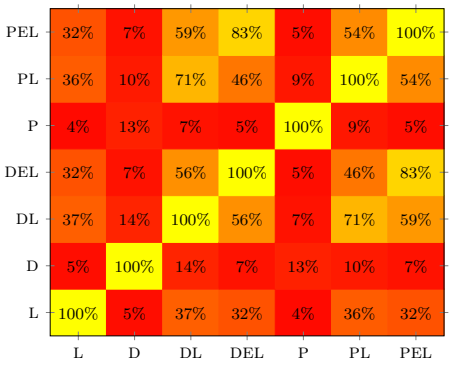


Fig. 5. Comparison of paths produced by different weights for $\mathcal{Q}_1$ over FULL

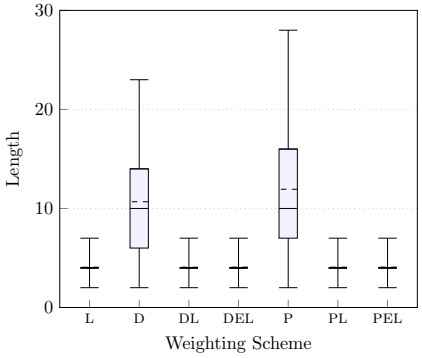


Fig. 6. Path lengths of different weighting schemes for for $\mathcal{Q}_1$ over FULL

see that DEL and PEL produce the same paths for 83% of the queries, with PL and DL coinciding for 71% of queries; however D and P have much lower overlap: 13%. To help explain this observation, in Figure 6 we present an overview of the lengths of paths produced by each scheme; notably D and P produce much longer paths on average since their weights are less affected by length, where longer paths are naturally much less likely to coincide. On the other hand, even though all *L measures tend to have short paths, we note that all weighted graphs tend to vary in their results from the baseline L (particularly D and P, which produce much longer paths). Finally, we observe a notable difference in DL and DEL (56% overlap) and between PL and PEL (54% overlap), suggesting that the introduction of edge weights significantly affects the paths computed.

6 User Study

In the previous section we discussed results on the performance of the path computation at different scales and for different weighting schemes; we also presented a statistical analysis of the differences between the paths produced by such schemes, noting that most schemes vary considerably, particularly from the baseline where only the length of the path is considered. In this section, we present a user study whose goal is to address our main research question:

- Do users prefer shortest paths in weighted versions of the graph (D*, P*) versus shortest paths by length only (L)?

Although a number of related datasets have been proposed for evaluating paths and entity relatedness (e.g., by Cheng et al. [8] and Talavera et al. [16]), these datasets do not provide a complete ranking of all paths between query nodes, which would of course be unfeasible given the number of potential simple paths between the nodes; for example, Cheng et al. [8] define a rather complex sequence of steps to reduce the number of paths; thereafter, to ease the burden on users, they adopt a pair-wise rating scheme where users are asked to choose which of two paths they prefer. Hence it was unclear how we could use such a dataset since our weighted shortest path framework only produces one path, which may not "hit" the partial/pairwise gold standard. Furthermore, the lengths of paths in existing datasets are bounded, whereas in our approach, per Figure 6, the D and P schemes can, in some cases, produce paths in excess of length 20. In the end, we decided to pursue a more direct strategy: ask users to score the paths produced by different weighting schemes for the same query pairs.

Queries: In the previous experiments we used randomly selected pairs of nodes; we deemed this set unsuitable for a user study since they referred to obscure entities, making it difficult for users to judge paths between them. Instead we manually created two sets of queries: the first set contains 10 pairs of entities of the same type, while the second set contains 10 pairs of entities of different types. We keep the number of pairs low to be able to generate numerous evaluations for the same pairs across different strategies. The nodes were chosen to reflect well-known entities that evaluators would likely be aware of such that they could better judge the generated paths. Importantly, the two query sets were selected before any paths were generated to avoid biasing the selection.

Scoring: To facilitate the evaluation of the paths, we created a version of the system that would display paths generated by all weighting schemes for a given query on one page. Instructions to users were minimal: they were asked to evaluate the path on a Likert scale with seven levels indicating how interesting the path was as an explanation for how the two entities were connected, scoring each generated path from 1 (worst), to 7 (best); other than that, scoring was left to the user's discretion. Displaying all paths for one query on the same page allowed users to consider paths produced by the other schemes when making their evaluation. The evaluators were 10 students from a Semantic Web course. To avoid long response times (see Figure 4), we used the 1.6 M dataset.

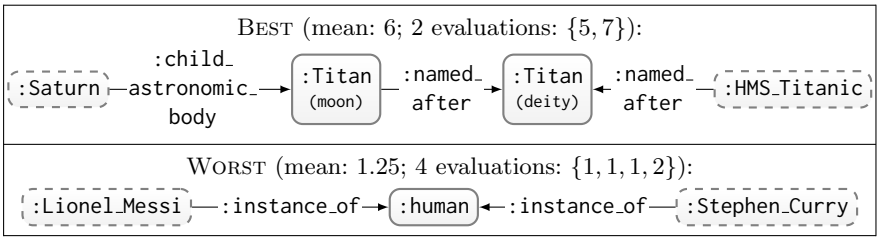


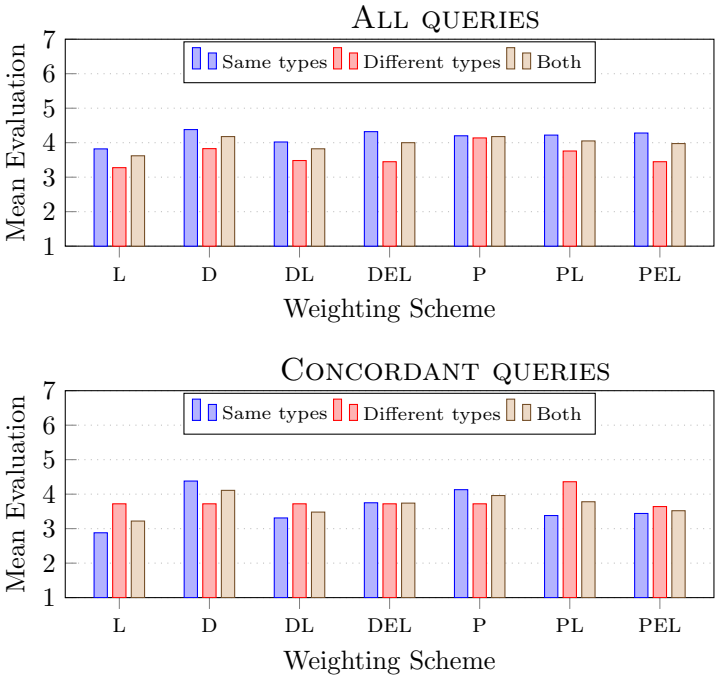
Fig. 7. Best and worst evaluated paths with multiple responses on a [1,7] scale

Evaluations: In total, 81 query evaluations were recorded, of which 79 were complete (2 incomplete evaluations were discarded). In total, we thus gathered an average of ∼4 evaluations for each query pair, totalling $79 \times 7 = 553$ scores for paths on a scale of 1–7. As an example, in Figure 7 we present the best (`:Saturn`,`:HMS_Titanic`) and worst (`:Lionel_Messi`,`:Stephen_Curry`) evaluated paths for which we received multiple evaluations.

Inter-rater agreement: Given that we use an ordinal scale for evaluations, we apply a correlation measure to assess inter-rater agreement: specifically we use Kendall's τ-b measure.[9] For each query pair, we measured the τ-b correlation between each distinct pair of evaluations across strategies, and then took the mean for each query; then taking the mean over all queries, the result was a macro-averaged correlation of $\tau = 0.201$, indicating positive but slight agreement. We believe this low correlation to be due to two main factors: (1) the subjective nature of the evaluation, where rather than providing specific instructions on how to rate paths (which may bias towards one strategy or another), we left users to decide their own criteria for rating; (2) we did not verify beforehand that interesting paths did indeed exist for the query entities, and so it may be the case that there is no clear "better" paths in some cases. To address this point, we consider two query sets for our results: the set of All queries as previously described; and a set of Concordant queries, intended to represent queries for which there was inter-rater consensus, defined as query pairs with multiple evaluations and a positive (τ-b) correlation; the latter sub-set contains 8 query pairs, with $\tau = 0.552$ and a total of $27 \times 7 = 189$ scores for individual paths.

Comparing evaluations of weighted shortest paths: In Figure 8, we present a high-level comparison of the mean evaluations of All queries and the subset of Concordant queries for different weighting schemes. We present these results considering (1) queries involving entities of the same type, (2) queries involving entities of different types, (3) the combination of both. As a general trend, we see that the paths tended to have mean evaluations tending towards the (neutral) middle of the Likert scale. Furthermore, we see that the mean evaluations in the

[9] Kendall's τ-b is a non-parametric rank coefficient measuring correlation between two lists of values in a range of $[-1, 1]$, where -1 indicates perfect inverse correlation, 0 no correlation, and 1 perfect correlation.



W.S.	p
D	0.022*
DL	0.230
DEL	0.026*
P	0.038*
PL	0.013*
PEL	0.034*

W.S.	p
D	0.006*
DL	0.182
DEL	0.020*
P	0.110
PL	0.016*
PEL	0.118

Fig. 8. Mean evaluations for each weighting scheme for the set of all queries and concordant queries; the table to the right of each plot indicates significance levels (paired Student t-test, 2-tailed, $n = 79$ for All, $n = 28$ for Concordant) for the difference of the mean between the baseline (L) and the given weighting scheme for all queries, where statistically significant results ($\alpha = 0.05$) are marked with '*'

Concordant set tend to be slightly lower than the All set, which suggests slightly more agreement on which paths are bad (per Figure 7). Returning to our main research question, the baseline without weights (L) had the worst mean evaluations in all results; to the right of Figure 8, we present the significance of the difference of the mean for each weighting scheme to the baseline, where for All we see that the mean evaluations of D, DEL, P, PL and PEL are significantly different ($\alpha = 0.05$) from the baseline; only the mean evaluation of the DL paths was not significantly different from that of L where, in Figure 5, we observed that DL has the most similar paths to L. For the Concordant set, though fewer evaluations are available ($n = 27$) significant results were still achieved for D, DEL, and PL.[10] In summary, though the hybrid weighting schemes we propose do not show any clear benefit to those based solely on node centrality, the results

[10] An interesting case is P, which achieved the second-highest mean, but did not reach significance: analysing the result further, P had a much lower Pearson correlation with L than other configurations, leading to the higher p-value.

show that users preferred weighted shortest paths over the unweighted baseline, with the results being significant in many cases.

Materials and Demo: We make code, queries, performance results, ratings, etc., available at https://github.com/GTartari/Weighted-Shortest-Paths. A demo over a subset of Wikidata (1.6 M) is available at http://wisp.dcc.uchile.cl/.

7 Conclusions

In this paper, we proposed to apply weightings to RDF graphs such that the weighted shortest paths produced are of more interest to the user than shortest paths based on length alone. The goal of such an approach is to enable the application of best-first search algorithms – such as Dijkstra's algorithm – to directly find and output the paths of most interest, avoiding fixed bounds. We then proposed a number of weighting schemes for RDF graphs based on path length, node centrality, and edge-label frequency, as well as combinations thereof.

With an efficient implementation based on Dijkstra's algorithm layered on top of in-memory graph traversal indexes, we showed that the approach can scale to the full Wikidata graph (25 million nodes, 90 million edges); however, response times increase linearly with scale, where queries on the full dataset (which must visit 12.3–14.5 million nodes) can take in the order of a minute; on the other hand, for a sub-graph of 1.2 million nodes and 6.6 million edges, queries can be executed in the order of 2–4 seconds. We further noted that the proposed weighting schemes can increase mean response times, requiring more nodes to be visited. We also observed that most weighting schemes produce largely disjoint results, though we found large overlap between the DL–PL and DEL–PEL schemes.

Finally, we presented a user study over Wikidata designed to address the main research question: are the weighted shortest paths of more interest to a typical user than the baseline shortest paths considering only length? We found that the user scoring of paths in general exhibited a low, positive inter-rater correlation, implying that the task of scoring paths is, in general, highly subjective; however, there was a positive consensus for a subset of the chosen queries. Thereafter we found that all weighting schemes outperformed the baseline in this evaluation, and that the results were significant ($\alpha = 0.05$) in most cases. This contributes evidence to validate our main hypothesis that appropriate weighting schemes can improve the perceived relevance of shortest paths for users.

Future work: There are many important directions left to be explored. A limitation of our current work is the performance over large graphs: while we see it as a positive to be able to scale to the full Wikidata graph, response times in the order of a minute would preclude interactive response times and test the patience of users; we believe that the best way to tackle this issue is with approximations, or perhaps using an appropriate heuristic with A*-search. Another limitation is that our methods currently only produce one path, though adaptations of Dijkstra-style algorithms do exists for top-k paths, including in an RDF setting [12, 13, 17]; such techniques can be adapted to weighted graphs.

Another area for improvement is with respect to evaluation: trying to evaluate which paths are of interest to users is challenging given the subjective nature of the task and the number of possible paths to choose from, precluding the creation of a reliable and complete gold standard [8]. This remains an open problem. On the other hand, while our current evaluation deals with paths between entities of general interest in the Wikidata graph, it would be interesting to explore domain-specific use-cases and graphs, such as coauthorship paths, protein sequencing, etc.; a main issue here would be to find domain experts capable of providing a meaningful evaluation of the paths suggested over specialist datasets.

Acknowledgements The work was supported by the Millennium Institute for Foundational Research on Data (IMFD) and by Fondecyt Grant No. 1181896.

References

1. Aleman-Meza, B., Halaschek-Wiener, C., Arpinar, I.B., Ramakrishnan, C., Sheth, A.P.: Ranking complex relationships on the Semantic Web. IEEE Internet Computing 9(3), 37–44 (2005)
2. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J.L., Vrgoc, D.: Foundations of Modern Query Languages for Graph Databases. ACM Comput. Surv. 50(5), 68:1–68:40 (2017)
3. Anyanwu, K., Maduko, A., Sheth, A.P.: SemRank: ranking complex relationship search results on the Semantic Web. In: World Wide Web International Conference (WWW). pp. 117–127 (2005)
4. Bianchi, F., Palmonari, M., Cremaschi, M., Fersini, E.: Actively learning to rank semantic associations for personalized contextual exploration of knowledge graphs. In: The Semantic Web (ESWC). pp. 120–135 (2017)
5. Butt, A.S., Haller, A., Xie, L.: DWRank: Learning concept ranking for ontology search. Semantic Web 7(4), 447–461 (2016)
6. Cedeño, J.P., Candan, K.S.: R^2DF framework for ranked path queries over weighted RDF graphs. In: International Conference on Web Intelligence, Mining and Semantics (WIMS) (2011)
7. Chen, N., Prasanna, V.K.: Learning to Rank Complex Semantic Relationships. Int. J. Semantic Web Inf. Syst. 8(4), 1–19 (2012)
8. Cheng, G., Shao, F., Qu, Y.: An Empirical Evaluation of Techniques for Ranking Semantic Associations. IEEE Trans. Knowl. Data Eng. 29(11), 2388–2401 (2017)
9. Cheng, G., Zhang, Y., Qu, Y.: Explass: Exploring associations between entities via top-k ontological patterns and facets. In: International Semantic Web Conference (ISWC). pp. 422–437 (2014)
10. Dijkstra, E.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
11. Farsani, H.K., Nematbakhsh, M.A., Lausen, G.: SRank: Shortest paths as distance between nodes of a graph with application to RDF clustering. J. Information Science 39(2), 198–210 (2013)
12. Filtz, E., Savenkov, V., Umbrich, J.: On finding the k shortest paths in RDF data. In: Workshop on Intelligent Exploration of Semantic Data (IESD) (2016)
13. Hassan, Z., Qadir, M.A., Islam, M.A., Shahzad, U., Akhter, N.: Modified MinG algorithm to find top-k shortest paths from large RDF graphs. In: SemWebEval Challenge at ESWC. pp. 213–227 (2016)

14. Heim, P., Hellmann, S., Lehmann, J., Lohmann, S., Stegemann, T.: RelFinder: Revealing relationships in RDF knowledge bases. In: International Conference on Semantic and Digital Media Technologies (SAMT). pp. 182–187 (2009)
15. Heim, P., Lohmann, S., Stegemann, T.: Interactive relationship discovery via the Semantic Web. In: 7th Extended Semantic Web Conference. pp. 303–317 (2010)
16. Herrera, J.E.T., Casanova, M.A., Nunes, B.P., Leme, L.A.P.P., Lopes, G.R.: An entity relatedness test dataset. In: International Semantic Web Conference (ISWC). pp. 193–201 (2017)
17. Hertling, S., Schröder, M., Jilek, C., Dengel, A.: Top-k shortest paths in directed labeled multigraphs. In: SemWebEval Challenge at ESWC. pp. 200–212 (2016)
18. Hulpuş, I., Prangnawarat, N., Hayes, C.: Path-Based Semantic Relatedness on Linked Data and Its Use to Word and Entity Disambiguation. In: International Semantic Web Conference (ISWC). pp. 442–457 (2015)
19. Jämsen, J., Niemi, T., Järvelin, K.: Derived types in semantic association discovery. J. Intell. Inf. Syst. 35(2), 213–244 (2010)
20. Karger, D.R.: The Semantic Web and end users: What's wrong and how to fix it. IEEE Internet Computing 18(6), 64–70 (2014)
21. Lehmann, J., Schüppel, J., Auer, S.: Discovering unknown connections – the DBpedia relationship finder. In: Social Semantic Web (CSSW). pp. 99–110 (2007)
22. Nunes, B.P., Herrera, J.E.T., Taibi, D., Lopes, G.R., Casanova, M.A., Dietze, S.: SCS connector – quantifying and visualising semantic paths between entity pairs. In: The Semantic Web (ESWC) Satellite Events. pp. 461–466 (2014)
23. Partl, C., Gratzl, S., Streit, M., Wassermann, A.M., Pfister, H., Schmalstieg, D., Lex, A.: Pathfinder: Visual Analysis of Paths in Graphs. Comput. Graph. Forum 35(3), 71–80 (2016)
24. Pirrò, G.: Explaining and Suggesting Relatedness in Knowledge Graphs. In: International Semantic Web Conference (ISWC). pp. 622–639 (2015)
25. Rusu, D., Fortuna, B., Mladenic, D.: Measuring concept similarity in ontologies using weighted concept paths. Applied Ontology 9(1), 65–95 (2014)
26. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM J. Comput. 8(3), 410–421 (1979)
27. Vidal, M., Rivera, J.C., Ibáñez, L.D., Raschid, L., Palma, G., Rodríguez-Drumond, H., Ruckhaus, E.: An authority-flow based ranking approach to discover potential novel associations between Linked Data. Semantic Web 5(1), 23–46 (2014)
28. Viswanathan, V., Krishnamurthi, I.: Ranking semantic relationships between two entities using personalization in context specification. Inf. Sci. 207, 35–49 (2012)
29. Viswanathan, V., Krishnamurthi, I.: Finding relevant semantic association paths using semantic ant colony optimization algorithm. Soft Comput. 19(1), 251–260 (2015)
30. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM 57(10), 78–85 (2014)
31. Zhang, Y., Cheng, G., Qu, Y.: Towards exploratory relationship search: A clustering-based approach. In: Joint International Conference on Semantic Technology (JIST). pp. 277–293 (2013)