# Learning to Assess Linked Data Relationships Using Genetic Programming

Ilaria Tiddi, Mathieu d'Aquin, Enrico Motta

Knowledge Media Institute
The Open University, United Kingdom
{ilaria.tiddi, mathieu.daquin, enrico.motta}@open.ac.uk

**Abstract.** The goal of this work is to learn a measure supporting the detection of strong relationships between Linked Data entities. Such relationships can be represented as paths of entities and properties, and can be obtained through a blind graph search process traversing Linked Data. The challenge here is therefore the design of a cost-function that is able to detect the strongest relationship between two given entities, by objectively assessing the value of a given path. To achieve this, we use a Genetic Programming approach in a supervised learning method to generate path evaluation functions that compare well with human evaluations. We show how such a cost-function can be generated only using basic topological features of the nodes of the paths as they are being traversed (i.e. without knowledge of the whole graph), and how it can be improved through introducing a very small amount of knowledge about the vocabularies of the properties that connect nodes in the graph.

## 1 Introduction

The goal of the work here presented is to automatically discover what makes a strong relationship between two entities of the Web of Linked Data. Identifying the strength of the relationship between entities can have many applications, the most common of which is to measure entity relatedness, i.e. identifying how related two entities are. This is a well-known problem for a wide range of tasks, such as text-mining and named-entity disambiguation in Natural Language Processing, or ontology population and query expansion in Semantic Web activities.

From a Web of Data perspective, a relationship can be identified in the graph of Linked Data as a semantic path (expressed as a chain of entities and properties) between two given entities, and graph search techniques can be used to reveal them. When applying such techniques to the Linked Data graph, however, the entities and properties included in the found paths might come from a number of different, unknown data sources. In order to avoid having to index and locally pre-process a necessarily partial subset of the graph, a natural approach is to rely on link traversal, which allows to incrementally and agnostically explore the graph from entity to entity until paths between them are found. In other words, finding relationships between entities in the Linked Data graph requires a uniformed (or blind) search, which does not need pre-computation or knowledge over the entire graph. However, to drive such an uniformed search, a function to measure the strength of the explored paths (a “cost-function”) is necessary to ensure that only the most promising ones will be followed.

Our goal is therefore to figure out which of (and how) the features of the Linked Data graph along the explored paths could be used by such cost-function. While one could intuitively think that the shortest paths reveal the strongest connections, this assumption does not necessarily hold within the Linked Data space, where entities of different datasets are connected by multiple paths of similar lengths. Our challenge is to find which Linked Data structural information we need in order to design a cost-function that objectively assesses the value of a path. More specifically, we aim at discovering which topological and semantic features of the traversed entities and properties can be used to reveal the strongest relationships.

To answer this question, the approach we propose is to use a supervised method based on Genetic Programming whose scope is to learn the path evaluation function to integrate in a Linked Data blind search. Our idea is that, starting from a randomly generated population of cost-functions created from a set of topological and semantic characteristics of the Linked Data graph, the evolutionary algorithm will reveal which functions best compare with human evaluations, and will show us what is really important to assess strong relationships in Linked Data. The learnt cost-functions are compared and discussed in our experiments, where we show not only that good results are achieved using basic topological features of the nodes of the paths as they are being traversed, but also how those results can be improved through introducing a very small amount of knowledge about the vocabularies used to label the edges connecting the nodes.

## 2 Related Work

As already mentioned, the goal of our work is to learn a measure to assess strong Linked Data relationships, so that this can be integrated in an uninformed graph search within Linked Data. For this reason, we divided this related work section in three parts. First, we study works that focus on assessing Linked Data entity relatedness, in order to discover which types of interestingness measures have been proposed. Then, we analyse works based on Linked Data traversal, to see how uninformed graph searches can be applied in the context of Linked Data. Finally, we explore works that focused on designing a measure empirically, namely by learning it through Genetic Programming.

**Linked Data Entity Relatedness.** There is a solid body of literature on entity relatedness, which can be categorised according to the corpus used to assess it [10, 18]. Here, we focus mainly of approaches that compute the relatedness based on Linked Data. A first area comprehends Linked Data-based metrics to assess the strength of a relationship between entities quantitatively [10, 14, 20, 21, 24]. They can be divided into entity-based approaches, which compute the similarity between neighbouring concepts based on the entity description (i.e.

triples where the entity is involved as a subject or object) [14, 21, 24], and path-based metrics [10, 20], which compute relatedness between concepts that are not directly connected. As in our work, these approaches are motivated by the idea of exploiting the rich resource descriptions existing in (and across) Linked Data. From our perspective, these works are too restrictive for two main reasons: first, they present ad-hoc measures, which have been either manually designed based on the analysed datasets or adapted from existing information theoretical measures; second, the strength of a relationship can only be assessed quantitatively.

A second area includes works that define entity relationships qualitatively, as Linked Data paths or subgraphs. The strongest relationships are identified through information theoretical measures based on node centrality, node frequency or edge informativeness applied on the paths retrieved from one or more Linked Data datasets. We find in this category systems for data visualisation and exploratory searches, such as RelFinder [8], REX [5], Explass [1] and, more recently, Recap [18]. These approaches first identify all possible relationships between two entities, either using SPARQL queries aiming at retrieving paths up to a certain length [8, 18], or by extracting them from a pre-computed dataset [1, 5], and then rank the results based on some predefined interestingness measures. Pathfinding techniques have also been used to identify entity relationships [3, 12, 15]: similar to our work is the use of cost-functions based on the Linked Data graph structure to drive the informed searches (e.g. A*, random walks), prioritising nodes and pruning the search space. With that said, their major limitation consist in exploiting Linked Data with an a priori knowledge, either by indexing and pre-processing datasets, or by using queries against SPARQL endpoints, therefore pre-defining the desired portion of data to be analysed.

**Linked Data Traversal.** The idea behind the Linked Data Traversal (link traversal in short) is to exploit URI dereferencing[1] to discover connections between entities across datasets on-the-fly and in a *follow-your-nose* fashion, so that no or very little domain knowledge has to be introduced. Link traversal relies on the fact that if data are connected (through owl:sameAs, skos:exactMatch, rdfs:seeAlso or simply by vocabulary reuse), then one can naturally span data-sources and gather new knowledge serendipitously. Various studies have shown that Linked Data can be traversed agnostically in contexts such as SPARQL query extensions [7] or (cor-)relation explanation [18, 22]. So far, however, uninformed graph searches have been only used for Linked Data crawling or indexing [9, 20]. To the best of our knowledge, no work has focused on identifying a cost-function suitable to be integrated in an uninformed search over Linked Data.

**Genetic Programming.** Evolutionary algorithms have proven to perform well in those tasks where it was necessary to identify suitable functions based on a desired output. For instance, Genetic Programming has been successfully applied in Information Retrieval to reveal the most appropriate document ranking functions for search engines [2, 4, 13, 23]. In the Linked Data context, Genetic

[1] Retrieving a representation of a resource identified by a URI

Programming was used to identify similarity functions for discovering links [16, 17], instance clustering [6] or matching [11] across different datasets, but not, prior to the work presented here, to assess relationship strength between Linked Data entities.

## 3 Motivation, Challenges and Approach

Our motivating scenario is a uniform-cost search process which traverses Linked Data on-the-fly with the aim of identifying the best relationship given two input entities. Uniform-cost search (*ucs*) is an uninformed, non-greedy best-first search strategy where a cost-function $g(n)$ chooses the next node to expand based on the cumulative cost of the edges from the start to that node $n$. When expanded, new children are created and stored in the queue accordingly. Since the nodes are generated iteratively, *ucs* does not require holding the whole graph in memory, which makes it suitable for large graphs; secondly, $g(n)$ being cumulative (i.e. paths never get shorter as the nodes are added), the search expands nodes in the order of their optimal path, which guarantees search optimality. These characteristics make *ucs* particularly suitable to a context such as Linked Data.

Our process is designed as a bi-directional search, whose aim is to find the path $p = \langle n^l \dots n^r \rangle$ that best represents the relation between two entities $n^l$ and $n^r$. By "best", it is intended that the relatedness between $n^l$ and $n^r$, expressed as a score assigned by the cost-function, is maximised. Because *ucs* does guarantee optimality, but its bi-directional version does not, we use a maximum number of node expansions to perform as a termination criterion. An example of such a process, showing how different structural information might be needed to find the best relationship, is presented below. Here, we used entities of the same dataset for clarity purposes but, as demonstrated by the experiments, the process can be applied on entities of two arbitrary datasets.

### 3.1 Example Scenario

Let us imagine that we want to identify the strongest relationship between two DBpedia entities, e.g. $n_1$=db:ASongOfIceAndFire(novel) and $n_2$=db:GOT-TV-series(episodes) of Figure 1. The process consists in:

(1) *Bi-directional search.* Given the two nodes $n^l$ and $n^r$, two uniform-cost searches $ucs^l$ and $ucs^r$ are performed simultaneously. Their objective is to iteratively build two search spaces, a left-directed one from $n^l$ and a right-directed one from $n^r$, to find a common node $n^c$.

(2) *Entity dereferencing.* Each search space is expanded by dereferencing the entity labelling the next node in the queue, and by finding all the entities that are linked to it. We do consider as "link" any edge of the node, i.e. both incoming and outgoing RDF properties of the dereferenced entity. In the example, $n_1$ is linked to 5 entities and $n_2$ to 4. As said, nodes are queued and dereferenced according to their cumulative cost from the start node $n^j$ (with $j \in \{l, r\}$), which guarantees optimality to both $ucs^j$. This step is repeated until one or more common nodes $n^c$ are found.

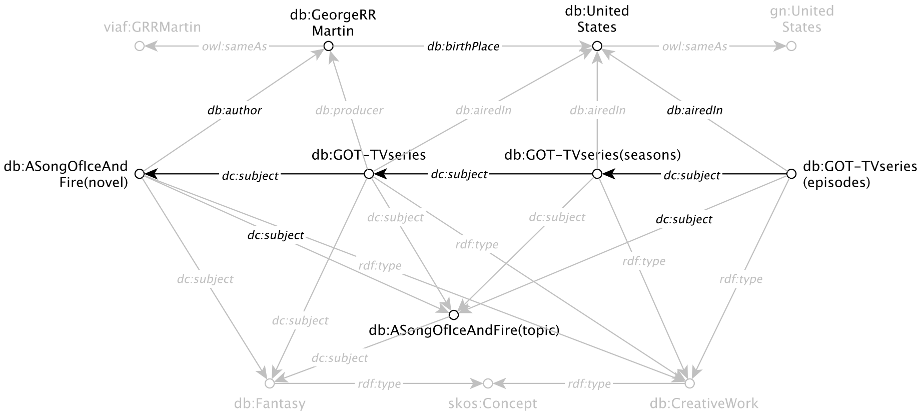

**Fig. 1.** Paths between db:ASongOfIceAndFire(novel) and db:GOT-TVseries(episodes).

(3) *Path building.* For each common node $n^c$, we build the two subpaths $p^j = \langle n^j \ldots n^c \rangle$, and then merge them into a path $p = \langle n^l \ldots n^c \ldots n^r \rangle$. Each path then identifies a relationship between the initial two entities. For instance, the graph of Figure 1 represents all the paths existing between $n_1$ and $n_2$ after a few iterations.

(4) *Path scoring.* The cost of each path is evaluated as an approximation (most often, a sum) of the costs of the paths from $n^l$ to $n^c$ and from $n^r$ to $n^c$. The one with the highest score, highlighted in the Figure, is chosen as the strongest relationship between the initial entities.

### 3.2 Challenge and Proposed Approach

From the process described above, it becomes clear that the problem to tackle are how to find a good cost-function is necessary to choose among a set of alternative paths between two entities, and how to avoid computational efforts or inconclusive searches. The question arising here is what is the best strategy to find the most representative relationships, and if we can exploit the information in the Web of Data to guide the two searches in Linked Data in the right direction, so that they can quickly get to convergence. When looking at the paths in Figure 1, an interesting observation can be made: the node corresponding to the entity db:GameOfThrones-TVseries has a lower indegree, which is generally used to measure the authority (its “popularity”) of a node, when compared to other nodes, as the ones labelled as ASongOfIceAndFire(topic) or db:UnitedStates. This information could be used to rank nodes so that the path that best specifies the relation between $n_1$ and $n_2$ is soon revealed. In other words, the structural features of the graph could be a good insight to drive a blind search in Linked Data. Given this, our challenge is to answer the question: what makes a path strong? Which are the topological or semantic features of a node or an edge, which can be used when deciding if a path is better than another? To reformulate the problem: can we use the structure of the Linked

Data graph to assess relationship strengths?

Our proposition is to use a supervised Genetic Programming (GP) approach to identify the cost-function that best performs in ranking sets of alternative relationship paths. Starting from a random population of cost-functions created on a set of features related to possible topological or semantic features of the nodes and edges of the path, the evolutionary algorithm will learn the cost-function that best performs when compared to a benchmark of human-evaluated relationship paths. The choice of Genetic Programming over other supervised learning techniques (e.g. SVMs, Neural Networks, Linear Regression or learning-to-rank) is motivated by three main reasons: first, its results are not assessed by comparing directly the path scores, which are hardly comparable to the human rankings provided in the benchmark, but by assessing the adequacy of the cost-functions through a fitness function; secondly, these formulas are human-understandable, which means that they can be used to identify the structural features of Linked Data that matter for a successful search; finally, because they are understandable, they can be directly implemented in a graph search mechanism. Additionally, the GP learning process is flexible, so it allows us to easily refine parameters and impose new constraints on the fitness function, and it comfortably deals with wide search spaces, so we can study large populations of possible cost-functions without worrying about scalability issues.

The contributions of this work can be summarised as follows: (i) we present a measure to detect strong entity relationships that can be integrated in uninformed searches over Linked Data, therefore avoiding data pre-processing; (ii) we demonstrate that such function can be derived empirically, which improves over the state-of-the-art approaches presenting domain-specific or manually-defined measures; (iii) we show that good results are achieved using basic topological features of the nodes of the paths as they are being traversed, and how those results can be improved through introducing a very small amount of knowledge about the vocabularies used to label the edges connecting the nodes.

# 4 Learning Functions to evaluate Paths

In this section, we first give and overview of the Genetic Programming framework and then present the supervised approach that we propose to discover the cost-functions to assess entity relationships.

## 4.1 Genetic Programming Foundations

Inspired by Darwin's theory of evolution, Genetic Programming is an Artificial Intelligence technique that aims at automatically solving problems in which the solution is not known in advance [19]. The general idea is to create a random population of computer programs, which are the candidate solutions for a problem, that the algorithm stochastic transforms ("evolves") into new, possibly improved, programs. The stochastic process guarantees that the GP proposes diverse solutions to a given problem.

In GP, programs are generally represented as trees of primitive elements, where the internal nodes (mathematical or logical operations) are called functions, while the leaf nodes (constants or variables) are called terminals. A fitness function measures how good each program is with respect to the problem to be solved. Given a population, a new population is created by adding programs using one of the three following genetic operations: (1) reproduction, in which a new child program is generated by copying a randomly selected parent program; (2) crossover, where a child program is generated by combining randomly chosen parts from two randomly selected parent programs; (3) mutation, where a new child program is generated by randomly altering a randomly chosen part of a selected parent. This process is iterated until a termination condition is met: typically either a maximum number of generations is reached, or a satisfying, possibly optimal solution (i.e. a desired fitness) is found. Along with the primitive set, the fitness and the termination condition, a set of parameters such as the population size, the probabilities of performing the genetic operations, the selection methodology or the maximum size for programs need to be decided to control the GP process.

### 4.2 Preparatory Steps

The described framework can be used to learn the cost-function that best ranks a set of alternative paths between two Linked Data entities. For a better understanding, we invite the reader to use as a reference the graph of Figure 1 and the three following paths:

$$\overline{p_1} = \text{db:ASongOfIce AndFire(novel)} \xrightarrow{\textit{db:author}} \text{db:George-RRMartin} \xrightarrow{\textit{db:birthPlace}} \text{db:United States} \xleftarrow{\textit{db:airedIn}} \text{db:GOT-TV series(episodes)}$$

$$\overline{p_2} = \text{db:ASongOfIce AndFire(novel)} \xleftarrow{\textit{dc:subject}} \text{db:GOT-TVseries} \xleftarrow{\textit{dc:subject}} \text{db:GOT-TV series(seasons)} \xleftarrow{\textit{dc:subject}} \text{db:GOT-TV series(episodes)}$$

$$\overline{p_3} = \text{db:ASongOfIce AndFire(novel)} \xrightarrow{\textit{dc:subject}} \text{db:ASongOfIce AndFire(topic)} \xleftarrow{\textit{dc:subject}} \text{db:GOT-TV series(episodes)}$$

**Process.** Let $P_i = \{\overline{p_1}, \ldots, \overline{p}_{|P_i|}\}$ be the set of $|P_i|$ alternative paths between two Linked Data entities, with $i$ being the $i$th pair in $D = \{P_1, \ldots, P_{|D|}\}$, the set of $|D|$ examples that have been ranked by humans, and $G = \{g_1, \ldots g_{|G|}\}$ a starting population of randomly generated cost-functions $g_j$. The GP algorithm iteratively evolves the population into a new, possibly improved one, until the stopping condition is met. The evolution consists first in assigning a fitness score to the cost-functions, which in our case reflects how "good" a cost-function is in ranking paths compared to the human evaluators. For instance, assuming 3 users have agreed on ranking the paths as $\overline{p_2}$, $\overline{p_3}$ and $\overline{p_1}$, those functions scoring the them in the same order will obtain the highest fitness. Then, reproduction, mutation and crossover are applied to some randomly (with bias from fitness) chosen individuals, and the generated children are added to the new population. The current population is replaced by the evolved one once they reach the same size, and a new generation starts.

**Primitives.** Terminals and functions are called primitives. A terminal can be: (i) a constant, i.e. a randomly chosen integer in the set $Z = \{0, \ldots, 1000\}$, or

(ii) a combination of an edge weighting function $w(e)$ (with $e$ being the edge) and one aggregator $a$. We call this combination $a.w$ an aggregated terminal.

Edge weighting functions $w(e)$ assign a weight to each edge of the path, based on the information of its source. We define 10 edge weighting functions, that we divide in topological and semantic terminals. Topological terminals focus on the Linked Data graph structure, and are as follows.

- *Fixed Weight* (1): the edge is assigned a score of 1. This is equivalent to performing a breadth-first search, where nodes are queued and explored in the order they are found.
- *Indegree* (*in*): the edge is weighted according to the number of incoming links of its source. For instance, the edge *db:birthPlace*(db:GeorgeRRMartin, db:UnitedStates) of Figure 1 has a weight of 2, since the source db:GeorgeRRMartin has 2 incoming links. This feature is chosen to understand the importance of "authority" nodes, i.e. the ones with many incoming links.
- *Outdegree* (*ou*): the edge is weighted according to the number of outgoing links of its source, e.g. the weight in the previous example is 2. *ou* helps us study the importance of "hub" nodes that point to many other nodes.
- *Degree* (*dg*): an edge is weighted based on the degree of the source, i.e. the sum of *in* and *ou*. To the previous example, *dg* would assign a score of 4.
- *Conditional Degree* (*cd*): the weight attributed to the edge depends on the RDF triple from which the edge has been generated. In fact, each edge $e(u, v)$ is generated from a dereferenced RDF triple, either $\langle u, e, v \rangle$, as in the case of *db:birthPlace*(db:GeorgeRRMartin, db:UnitedStates), or $\langle v, e, u \rangle$, as for *db:producer*(db:GeorgeRRMartin, db:GOT-TVseries). The *cd* terminal returns either the indegree or the outdegree of the source depending on whether the triple represents a back or a forward link. Therefore, *cd* would return 2 in the former case (the indegree of the node for db:GeorgeRRMartin) and 2 in the latter case (its outdegree). The conditional degree analyses the importance of paths going through large hubs, that are also common to many other paths.

We define semantic terminals those features that are more specific to Linked Data than to common graphs. For that, we first considered the vocabulary usage, then analysed the most frequent RDF properties, as provided by both Linked Open Vocabularies[2] and LODStats[3]. Note that, since we rely upon entity dereferencing to traverse Linked Data, we only considered the most frequent object properties.

- *Namespace Variety* (*ns*): an edge is weighted depending on the number of namespaces of its source node. For instance, the node db:GeorgeRRMartin has the two namespaces *owl:* and *db:* for its three links, while the node db:GOT-TVseries has the 3 namespaces *dc:*, *db:* and *skos:* for its 5 links. Namespaces variety is intended to analyse the use of vocabularies when semantically describing an entity. While initially we considered incoming and outgoing namespaces separately, we did not find any substantial difference in the process, and eventually reduced the two terminals to one.

[2] http://lov.okfn.org/dataset/lov/terms

[3] http://lodstats.aksw.org/

- *Type Degree* (*td*): the edge weight depends on the number of *rdf:type* declared for the source entity. For example, db:ASongOfIceAndFire(novel) has a type degree of 1 but, assuming this was declared as a skos:Concept too, its score would be 2. *td* focuses on the taxonomical importance of an entity, with the idea that the more a node is generic (i.e. the entity belongs to many classes), the less informative the path might be. Since *rdf:type* is unidirectional, there is no need to distinguish between in- and outdegree.
- *Topic Outdegree* (*so*): the edge weight is assigned by counting the number of outgoing edges labeled as *dc:subject*, *foaf:primaryTopic* and *skos:broader* of the starting node. The edge *db:author*(db:ASongOfIceAndFire(novel), db:GeorgeRRMartin) has a score of 2. The topic outdegree focuses on authority nodes in topic taxonomies (controlled vocabularies or classification codes).
- *Topic Indegree* (*si*): similarly, the edge weight is assigned by counting the number of incoming *dc:subject*, *foaf:primaryTopic* and *skos:broader* edges. The same edge has a score of 1 in this case. *si* considers hub nodes on controlled vocabularies.
- *Node Equality* (*sa*): the edge is weighted according to how much its source is connected to the external datasets, based on the number of links labeled as *owl:sameAs*, *skos:exactMatch* or *rdf:seeAlso*. For instance, db:UnitedStates is connected to its Geonames[4] corresponding entity gn:6252001 so, according to the *sa* weight, the edge *db:airedIn*(db:UnitedStates, db:GOT-TVseries(episodes)) is scored 1. *sa* considers the importance of the inter-dataset connections. Since those properties are bi-directional, we do not distinguish between in- and outdegree.

Aggregators are functions to combine the weights of edges across the whole path: *sum* returns the sum of the $w(e)$ for each of the $l$ edges of the path; *avg* returns the average edge weight across the path; *min* and *max* the path minimal an maximal $w(e)$, respectively.

To generate an individual, the aggregated terminals are randomly combined through the GP function set, composed of *addition* $x + y$, *multiplication* $x * y$, *division* $x/y$ and *logarithm* $log(x)$. For example, $g_1 = sum.1 + (1/avg.td)$ is interpreted as a function acting almost as a depth-first search, with a small added value from the average type degree of the nodes of the path.

**Fitness Evaluation.** The fitness of a cost-function is measured with the Normalised Discounted Cumulative Gain (*nDCG*), generally used in Information Retrieval to assess the quality of rankings provided by the web search engines based on the graded relevance of the returned documents[5]. The closer it gets to 1, the more the engine judges the documents as relevant as the human evaluators did. We apply the same idea by considering a path as a document, therefore evaluating first the *DCG* for a path $p_k$ at rank $k$ as:

$$DCG(p_k) = rel_1 + \sum_{m=2}^{k} \frac{rel_m}{log_2(m)} \tag{1}$$

[4] http://www.geonames.org/

[5] https://www.kaggle.com/wiki/NormalizedDiscountedCumulativeGain

where $rel_k$ is the average of the relevance scores given to $p_k$ by human evaluators. The $DCG(p_k)$ is then normalised by comparing it to its ideal score $iDCG(p_k)$, as assessed by the gold standard.

The function $avg(P_i)$ then averages each $nDCG(p_k)$ in the set $P_i$, so that to obtain the performance of the function for the $i$-th pair, as in Equation 2. The overall fitness of a function is obtained by averaging each $avg(P_i)$ of all the $|D|$ pairs of the dataset as in Equation 3.

$$avg(P_i) = \frac{\sum_{p_k \in P_i} nDCG(p_k)}{|P_i|} \qquad (2) \qquad\qquad f(g_j) = \frac{\sum_{P_i \in D} avg(P_i)}{|D|} \qquad (3)$$

We also add a penalty weight to avoid long and complex cost-functions, by comparing the length $l$ of a function with its ideal length $L$. The weighted fitness of a function is defined as:

$$fw(g_j) = f(g_j) - (w \times (l - L)^2) \qquad (4)$$

where $w$ is the penalty weight.

**Genetic Operations.** We perform the following genetic operations.

- Reproduction. Given a cost-function parent, a new individual is copied in the new generation without alterations.
- Crossover. Given two parents, two children are generated by swapping two random subtrees of the parents.
- Mutation. Given a selected parent, one node (the mutation point $mp$) is modified. We designed different kinds of mutations, as in Table 1, depending on the type of $mp$: if it is a constant $x$, the node is mutated with a new constant $y$ that is either higher (1) or lower (2) than $x$ in the range of $y = [x - 100, x + 100]$; if $mp$ is an aggregated terminal, the node is mutated by either modifying its aggregator (3), modifying its edge weighting function (4), or by replacing it with a new constant (5); if $mp$ is a function, it can be replaced either with a new constant (7) an aggregated terminal (8), or with a new function (8), in which case we might remove (9) or add (10) a child depending on the arity of the new function.

**Table 1.** Mutation examples for $g_1 = sum.1 + (1/avg.td)$.

| n. | $mp$ | mutation type | example |
|---|---|---|---|
| 1 | constant | $x < y < x + 100$ | $sum.1 + (\mathbf{18}/avg.td)$ |
| 2 | constant | $x - 100 < y < x$ | $sum.1 + (\mathbf{-18}/avg.td)$ |
| 3 | terminal | new $a$ | $sum.1 + (1/\mathbf{max}.td)$ |
| 4 | terminal | new $w$ | $sum.1 + (1/avg.\mathbf{in})$ |
| 5 | terminal | new $x$ | $sum.1 + (1/\mathbf{20})$ |
| 6 | function | new $a.w$ | $sum.1 + \mathbf{max.ns}$ |
| 7 | function | new $x$ | $sum.1 + \mathbf{40}$ |
| 8 | function | new function (same arity) | $sum.1 + (1 \times avg.td)$ |
| 9 | function | new function (delete child) | $\mathbf{log}(1/avg.td)$ |
| 10 | function | new function (add child) | $\mathbf{min.ou}\times(sum.1 + (1/avg.td))$ |

**Training and Testing.** We randomly split the dataset into a training set and a test set. Then, we run the GP process on the training set and store a small set of the fittest individuals, i.e. the cost-functions that performed better in ranking paths, while the rest are discarded. Third, the surviving individuals are tested on the test set, and if their fitness is not consistent with the one of the training set, we screen them out. This helps in avoiding overfitting and in obtaining more valid cost-functions. We then keep the best individual of each run.

## 5 Experiments

The section introduces our experimental scenario, describing the dataset we built and the control parameters for the Genetic Programming learning process. Then, it presents the obtained results, including the discovered cost-function[6].

### 5.1 Experimental Setting

As previously mentioned, the fitness is assessed on a dataset composed by sets of alternative paths between random pairs of entities. In order to create more variety in the final dataset, so that the learnt functions would not be overfitted to a specific type of data source, we used different types of entities, randomly extracted from different Linked Data sources, namely: (i) 12,630 *events* (from battles to sport to music events) from Yago[7]; (ii) 8,185 *people* from the Virtual International Authority File (VIAF)[8]; (iii) 999 *movies* from the Linked Movie Database[9]; and (iv) 1,174 *countries* and *capitals* from Geonames and the UNESCO[10] datasets.

To make sure to span at least to another dataset when finding paths, therefore guaranteeing more path heterogeneity, we used the DBpedia SPARQL endpoint as a pivot, i.e. we chose a desired `?_class` (event, country, person etc...) and the `?_dataset` we wanted to retrieve it from, and then ran the simple query:

```
select distinct ?same where {
 ?entity a ?_class. # select the entities of a desired class
 ?entity <http://www.w3.org/2002/07/owl#sameAs> ?same.  # get owl:sameAs
 FILTER(strStarts(str(?same), ?_dataset)).  # filter by dataset
}
ORDER BY RAND()  # make sure to get random entities
```

Next, given a random pair, we ran a bi-directional breadth-first search limited to 30 iterations in order to find a set of possible paths between them. Note that other iterations thresholds were also tested (between 20 and 50), and 30 cycles seemed the most reasonable trade-off between missing some relationships and taking more time to obtain almost the same relationships. We discarded the pairs for which no path was found. 8 judges were asked to evaluate each

[6] Dataset and results are available online at http://linkedu.eu/dedalo/pathfinding/

[7] http://yago-knowledge.org

[8] http://viaf.org/

[9] http://www.linkedmdb.org/

[10] http://uis.270a.info/.html

set, assigning the paths *rel* scores between 2 (“highly informative”) and 0 (“not informative at all”), and discarded the pairs whose agreement was below 0.1 according to the Fleiss’$k$ rating agreement[11]. The choice of using different scores was motivated to represent the gradation between meaningless relations, valide but weak relations and strong relations. An example of a path to be ranked, showing that the movie “The Skin Game” and the actress Dina Korzun are both based in Europe, is presented in Figure 2. The final dataset consisted of 100 pairs, whose paths were assigned a score corresponding to the average of the scores given by the users.

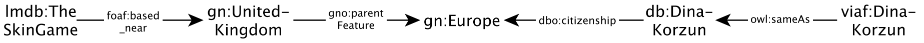

**Fig. 2.** A path example.

Finally, Table 2 presents the control parameters we used during the GP process. Because a perfect agreement between the functions with the users could not be reached, we use a maximum number of generations as a termination condition. In order to not bias the generation process, we also generated trees without limit in depth, and equally distributed the probability of functions, constants and aggregated terminals being chosen. It is worth mentioning that other parameters were also tested but, due to space limitation, we only present the ones giving the best results according to our tests.

**Table 2.** Control Parameters for the GP runs.

| | | | |
|---|---|---|---|
| Population size | 100 individuals | Reproduction | 10% population size |
| Max generations | 300 | Elitism | 10% population size |
| Termination | Max generation | Penalty weight $w$ | 0.001 |
| Selection | 5-sized tournament | Ideal length $L$ | 3 |
| Crossover rate: | 0.65 | Validation split | 70% – 30% |
| Mutation rate: | 0.15 | Num. individuals (testing) | 5 |

### 5.2 Results

First, we present the results of different runs of the Genetic Programming learning process presented in the previous Section. Table 3 shows the unweighted fitness on training set ($f_{tr}$) and test set ($f_{ts}$) of the best cost-functions learnt during 3 different runs (therefore different dataset cuts). We divided the GP runs depending on whether we used topological terminals only (T), or both topological and semantic terminals (S).

Results show that some terminals, i.e. the conditional degree *cd*, the namespace variety *ns* and the topic indegree *si*, are recurrent across different runs of the same block, which demonstrate the stability of our learning process. Given the regularity we noticed of the *min.ns* aggregated terminal, Table 3 also includes a third block of experiments (N), in which we used only the topological terminals and *min.ns*. We observe that both the T- and the N-functions, based

[11] https://en.wikipedia.org/wiki/Fleiss%27_kappa

**Table 3.** Best cost-functions for different runs.

| Run | Fittest individual $g_j$ | $f_{tr}$ | $f_{ts}$ |
|---|---|---|---|
| $T_1$ | $log(log(min.cd \times min.cd))/max.cd$ | 0.79 | 0.79 |
| $T_2$ | $log(min.cd)/(avg.cd + 87)$ | 0.77 | 0.78 |
| $T_3$ | $min.cd \times (min.cd/max.cd)$ | 0.78 | 0.72 |
| $N_1$ | $(log((max.ns/max.cd))/avg.ns) + min.ns$ | 0.82 | 0.81 |
| $N_2$ | $((min.dg/sum.cd)/sum.ou) + min.ns$ | 0.79 | 0.77 |
| $N_3$ | $min.ns/(log(max.cd)/avg.ns)$ | 0.83 | 0.75 |
| $S_1$ | $min.ns + (sum.ns/log(log(sum.si)))$ | 0.88 | 0.83 |
| $S_2$ | $min.ns + (min.cd/log(log(sum.si)))$ | 0.88 | 0.86 |
| $S_3$ | $min.ns + (log(max.in)/log(log(sum.si)))$ | 0.87 | 0.86 |

mostly on topological features, have a lower performance when compared to the S-ones, that include semantic terminals too. Nevertheless, the S-functions confirm the importance of $min.ns$.

We then performed a comparative evaluation between the learnt cost-functions and some related work of the literature, namely RelFinder (RF [8]), Recap [18] and the two measures presented by the Everything is connected Engine (EICE [3]) and by Moore et al. (M&V [15]). Figure 3 presents the $avg(P_i)$ score (Y axis) that each of the functions obtained on each of the examples in $D$ (X axis).

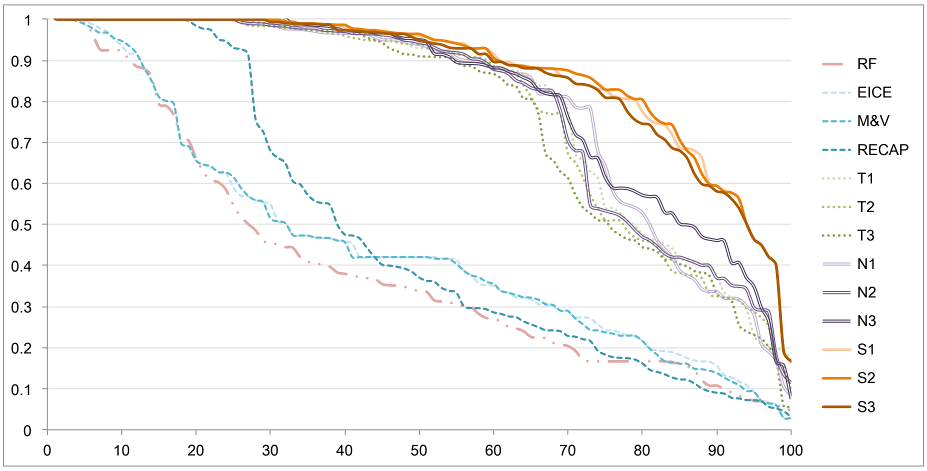

**Fig. 3.** $avg(P_i)$ of each measure on the full dataset $D$.

What can be noticed from the Figure is a considerable difference between the existing approaches, which are based on ad-hoc information theoretical measures, and the ones that were automatically learnt through Genetic Programming. Indeed, the combination of several topological characteristics sensibly improves a cost-function performance, as demonstrated by the overall fitness of $f(g_i)$ of $T_1$, $T_2$ and $T_3$ (also presented in Table 4, first row). This means that the ranking the T-functions give for a set of path is much more similar to the ones of a human evaluator than the ones attributed by hand-crafted measures. The low performance of the existing measures suggests that they are not suitable to correctly evaluate paths that connect entities across several Linked Data datasets, as the ones we have collected in our experiments. A slight improvement can also be observed with the N-functions: the overall fitness $f(g_i)$ for them improves roughly by 0.02-0.04 when compared to the T-functions. With that said, the Figure clearly shows that adding some semantic information is the key to obtain more precise results, as the S-function overall fitnesses $f(\mathrm{S}_1)$=0.86, $f(\mathrm{S}_2)$=0.88 and $f(\mathrm{S}_3)$=0.87 demonstrate (i.e. fitness improvement is ca. 0.09–0.11).

In Table 4, the cost-functions are compared to the baselines to assess if they perform in a stable way across different datasets sources. We removed, in turn, pairs whose entities belonged to one of the Linked Data sources presented in Section 5.1, and then calculated the functions' fitness $f(g_i)$ on the filtered dataset. Results confirm that the S-functions are consistent even with different datasets.

**Table 4.** Overall fitness of the functions across datasets. $D\backslash$ indicates from which dataset the entities were removed; $s$ indicates the size of the filtered dataset.

| $D\backslash$ | $s$ | RF | RECAP | EICE | M&V | $\mathrm{T}_1$ | $\mathrm{T}_2$ | $\mathrm{T}_3$ | $\mathrm{S}_1$ | $\mathrm{S}_2$ | $\mathrm{S}_3$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ∅ (tot) | 100 | 0.399 | 0.481 | 0.449 | 0.446 | 0.784 | 0.763 | 0.749 | 0.871 | **0.873** | 0.865 |
| Geonames | 23 | 0.455 | 0.457 | 0.584 | 0.605 | 0.756 | 0.710 | 0.641 | 0.909 | **0.911** | 0.887 |
| Yago | 77 | 0.371 | 0.513 | 0.432 | 0.424 | 0.819 | 0.820 | 0.819 | **0.883** | 0.881 | 0.880 |
| VIAF | 69 | 0.378 | 0.492 | 0.418 | 0.414 | 0.832 | 0.828 | 0.801 | 0.880 | **0.888** | 0.880 |
| Unesco | 79 | 0.390 | 0.457 | 0.442 | 0.437 | 0.784 | 0.764 | 0.750 | 0.858 | **0.860** | 0.849 |
| LMDB | 73 | 0.439 | 0.382 | 0.438 | 0.435 | 0.740 | 0.728 | 0.705 | 0.846 | **0.847** | 0.843 |

We finally analyse the cost-function that reported the best performance:

$$S_2 = min.ns + \frac{min.cd}{log(log(sum.si))} \tag{5}$$

and observe that the terminals here included are the same that we had already noted as being the most recurrent among the different runs of Table 3. As can be seen from its shape, this function prioritises paths that:

- pass through nodes with rich node descriptions (the higher *min.ns* is, the more relevant the path is considered);
- do not include high level entities (that have many incoming *dc:subject*/*foaf:primaryTopic*/*skos:broader* links, since many other entities are also of the same category), since the higher *sum.si* is, the lower the path score is;
- include only specific entities (not hubs) for paths with a small number of topic categories. Indeed, because of the use of the double *log* function, the ratio between *min.cd* and *log*(*log*(*sum.si*)) is negative if *sum.si* is lower than 10. However, *min.cd* becomes a positive factor when *sum.si* is above 10.

In other words, the function prioritises specific paths (e.g. a movie and a person are based in the same region) to more general paths (e.g. a movie and a person are based in the same country).

## 6 Conclusions

In this paper, we presented a supervised method based on Genetic Programming in which we learnt a measure to detect strong relationships between entities in the Linked Data graph. Such measure is a cost-function to be used in a blind graph search over Linked Data, in which relationships between entities are identified as Linked Data paths of entities and properties. With the assumption that the topological and semantic structure of Linked Data can be used by a cost-function to identify the strongest connections, we used Genetic Programming to generate a population of cost-functions that was evolved iteratively, based on how well the individuals compared with a human evaluated training data. The results proved our idea that successful path evaluation functions can be built empirically using basic topological features of the nodes traversed by the paths, and that a little knowledge about the vocabularies of the properties connecting nodes in the explored graph is fundamental to obtain the best cost-functions. We analysed the obtained functions to detect which features are important in Linked Data to find the strongest entity relationships, and finally presented the cost-function that we learnt. As future work, we will integrate this cost-function in a Linked Data pathfinding process that can be used in frameworks for on-the-fly knowledge discovery over Linked Data.

## References

1. Gong Cheng, Yanan Zhang, and Yuzhong Qu. Explass: Exploring associations between entities via top-k ontological patterns and facets. In *The Semantic Web–ISWC 2014*, pages 422–437. Springer, 2014.
2. Oscar Cordón, Enrique Herrera-Viedma, Cristina López-Pujalte, María Luque, and Carmen Zarco. A review on the application of evolutionary computation to information retrieval. *International Journal of Approximate Reasoning*, 34(2):241–264, 2003.
3. Laurens De Vocht, Sam Coppens, Ruben Verborgh, Miel Vander Sande, Erik Mannens, and Rik Van de Walle. Discovering meaningful connections between resources in the web of data. In *LDOW*, 2013.
4. Weiguo Fan, Michael D Gordon, and Praveen Pathak. A generic ranking function discovery framework by genetic programming for information retrieval. *Information Processing & Management*, 40(4):587–602, 2004.
5. Lujun Fang, Anish Das Sarma, Cong Yu, and Philip Bohannon. Rex: explaining relationships between entity pairs. *Proceedings of the VLDB Endowment*, 5(3):241–252, 2011.
6. Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. Metric-based stochastic conceptual clustering for ontologies. *Information Systems*, 34(8):792 – 806, 2009. Sixteenth {ACM} Conference on Information Knowledge and Management (CIKM 2007).
7. Olaf Hartig. Squin: a traversal based query execution system for the web of linked data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1081–1084. ACM, 2013.
8. Philipp Heim, Sebastian Hellmann, Jens Lehmann, Steffen Lohmann, and Timo Stegemann. Relfinder: Revealing relationships in rdf knowledge bases. In *Semantic Multimedia*, pages 182–187. Springer, 2009.

9. Aidan Hogan, Andreas Harth, Jürgen Umbrich, Sheila Kinsella, Axel Polleres, and Stefan Decker. Searching and browsing linked data with swse: The semantic web search engine. *Web semantics: science, services and agents on the world wide web*, 9(4):365–401, 2011.
10. Ioana Hulpuş, Narumol Prangnawarat, and Conor Hayes. Path-based semantic relatedness on linked data and its use to word and entity disambiguation. In *The Semantic Web-ISWC 2015*, pages 442–457. Springer, 2015.
11. Robert Isele and Christian Bizer. Learning linkage rules using genetic programming. In *Proceedings of the Sixth International Workshop on Ontology Matching*, pages 13–24, 2011.
12. Gjergji Kasneci, Shady Elbassuoni, and Gerhard Weikum. Ming: mining informative entity relationship subgraphs. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1653–1656. ACM, 2009.
13. Jung Yi Lin, Jen-Yuan Yeh, and Chao Chung Liu. Learning to rank for information retrieval using layered multi-population genetic programming. In *Computational Intelligence and Cybernetics (CyberneticsCom), 2012 IEEE International Conference on*, pages 45–49. IEEE, 2012.
14. Rouzbeh Meymandpour and Joseph G. Davis. *Web Technologies and Applications: 15th Asia-Pacific Web Conference, APWeb 2013, Sydney, Australia, April 4-6, 2013. Proceedings*, chapter Linked Data Informativeness, pages 629–637. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
15. Joshua L Moore, Florian Steinke, and Volker Tresp. A novel metric for information retrieval in semantic networks. In *The Semantic Web: ESWC 2011 Workshops*, pages 65–79. Springer, 2012.
16. Axel-Cyrille Ngonga Ngomo and Klaus Lyko. Eagle: Efficient active learning of link specifications using genetic programming. In *The Semantic Web: Research and Applications*, pages 149–163. Springer, 2012.
17. Andriy Nikolov, Mathieu d'Aquin, and Enrico Motta. Unsupervised learning of link discovery configuration. In *The Semantic Web: Research and Applications*, pages 119–133. Springer, 2012.
18. Giuseppe Pirrò. Explaining and suggesting relatedness in knowledge graphs. In *The Semantic Web-ISWC 2015*, pages 622–639. Springer, 2015.
19. Riccardo Poli, William B Langdon, Nicholas F McPhee, and John R Koza. *A field guide to Genetic Programming*. Lulu. com, 2008.
20. Michael Schuhmacher and Simone Paolo Ponzetto. Knowledge-based graph document modeling. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 543–552. ACM, 2014.
21. Stephan Seufert, Srikanta J Bedathur, Johannes Hoffart, Andrey Gubichev, and Klaus Berberich. Efficient computation of relationship-centrality in large entity-relationship graphs. In *International Semantic Web Conference (Posters & Demos)*, pages 265–268, 2013.
22. Ilaria Tiddi, Mathieu d'Aquin, and Enrico Motta. Dedalo: Looking for clusters explanations in a labyrinth of linked data. In *The Semantic Web: Trends and Challenges*, pages 333–348. Springer, 2014.
23. Ricardo da S Torres, Alexandre X Falcão, Marcos A Gonçalves, João P Papa, Baoping Zhang, Weiguo Fan, and Edward A Fox. A genetic programming framework for content-based image retrieval. *Pattern Recognition*, 42(2):283–292, 2009.
24. Wenlei Zhou, Haofen Wang, Jiansong Chao, Weinan Zhang, and Yong Yu. Loddo: Using linked open data description overlap to measure semantic relatedness between named entities. In *The Semantic Web*, pages 268–283. Springer, 2012.