Flexible Querying of Lifelong Learner Metadata

Alexandra Poulovassilis and Peter T. Wood

London Knowledge Lab, Birkbeck, University of London, UK
{ap,ptw}@dcs.bbk.ac.uk

Abstract. We propose combining query approximation and query relaxation techniques in order to support flexible querying of heterogeneous data arising from lifelong learners' educational and work experiences. A key aim of such querying facilities is to allow learners to identify possible choices for their future learning and professional development from what others have done. With our approach, query results can be computed incrementally, in polynomial time, and returned in order of increasing "distance" from the user's original query.

1 Introduction

Supporting the needs of lifelong learners is leading to research into new learner-centred models of delivering learning resources and opportunities [19, 18] and into the role of online support in providing careers guidance [3, 5]. In this direction, the L4All system aims to support lifelong learners in exploring learning opportunities and in planning and reflecting on their learning [6]. Consultation early in the L4All project with groups of Further Education (FE) and Higher Education (HE) learners found that careers advice is often patchy, especially at critical decision points, that social networks and factors influence educational and career choices, and that word-of-mouth recommendations play an important role. There is also a link between careers guidance and learner retention, based on correcting false assumptions and creating learners' expectations that are in line with the real-life experiences of others.

The L4All system allows users to create and maintain a chronological record of their learning, work and personal episodes — their *timelines*. This approach is distinctive in that the timeline provides a record of lifelong learning, rather than learning at just one stage or period, and provides us with a tool to understand social as well as educational factors that may influence career decisions and educational choices.

Figure 1 illustrates a fragment of the data and metadata relating to a user's timeline (where `sc` denotes `subclassOf`). The episodes within a timeline have a start and an end date associated with them (for simplicity, these are not shown in Figure 1). Episodes are ordered according to their start date — as indicated by edges labelled `next`. There are several types of episode, for example `University` and `Work` in our example. Associated with each type of episode are several properties — for simplicity, we have shown just two of these, `qualif[ication]` and `job`.

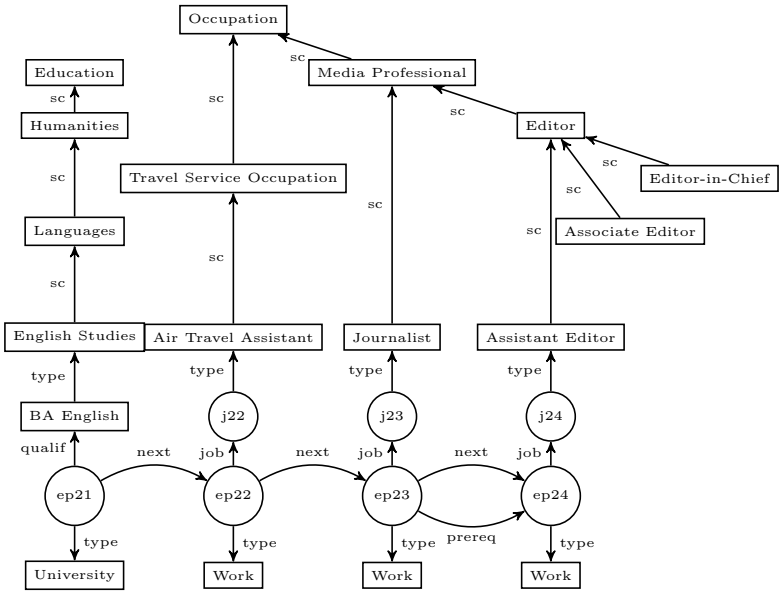

Fig. 1. A fragment of timeline data and metadata

A key aim of the L4All system is to allow learners to search over the timeline data, and to identify possible choices for their own future learning and professional development by seeing what others have done. In particular, [22, 21] describe a facility for searching for "people like me" which takes users through a three-step process in searching for timelines that may be relevant to them:
(i) The user specifies which attributes of their profile should be matched with other users' profiles, which acts as a filter of possible candidate timelines.
(ii) The user specifies which parts of their own timeline should be matched with other users' timelines, by selecting the required categories of episode. There are some 20 categories of episode; some categories are annotated with a primary and possibly a secondary classification, drawn from standard U.K. occupational and educational taxonomies (see Figure 1 for an indicative fragment of such metadata). Each classification hierarchy may have up to four levels.
(iii) The user selects the "depth" of episode classification that should be taken into account when matching their own timeline data with that of others, and selects the similarity metric that should be applied. The metric can be one of four alternatives: Jaccard Similarity, Dice Similarity, Euclidean Distance, and NeedlemanWunch Distance (see `www.dcs.shef.ac.uk/~sam/stringmetrics.html`).

Once the user's definition of "people like me" has been specified, the system returns a list of all the candidate timelines, ranked by their normalised similarity. The user can then select one of these timelines to visualise in detail.

Although this facility certainly helps users to find and explore relevant timelines and episodes, it is rather rigid for a number of reasons: it offers a fixed

set of similarity metrics over the timeline data; it allows just a single level of detail to be applied to the classifications of the selected categories of episode; and the similarity matching is applied to all these categories of episode in the user's timeline and the target timelines. Thus, there is limited flexibility for users to formulate their precise requirements for the timeline search, and to explore alternative formulations of selected parts of their query.

This paper proposes new flexible querying techniques for supporting users' search over the L4All database of timelines. We begin with a discussion of the background and motivation for our approach in Section 2. In Sections 3 and 4 we define the notion of *ApproxRelax queries*, which combine query *approximation* and query *relaxation* capabilities: we first define the semantics of single-conjunct ApproxRelax queries and we discuss the evaluation of such queries; we then define the semantics of general ApproxRelax queries, discuss their complexity, and present a polynomial-time algorithim for incrementally computing the top-k answers to such queries. We give our concluding remarks in Section 5.

2 Motivation

We assume here a general semistructured data model comprising a directed graph $G = (V, E)$. Each node in V is labelled with a constant and each edge is labelled with a symbol l drawn from a finite alphabet Σ. This general graph model encompasses RDF/S data, except that it does not allow for the representation of RDF's "blank" nodes. Also, in this paper we assume that only a limited fragment of the RDFS vocabulary is interpreted — `rdf:type` and `rdfs:subClassOf` — which we abbreviate by `type` and `sc`.

In [12], we considered *conjunctive regular path* (CRP) queries [2] over our general graph model (but without support for any interpreted vocabulary or query relaxation). Such queries are of the form

$$(Z_1, \ldots, Z_m) \leftarrow (X_1, R_1, Y_1), \ldots, (X_n, R_n, Y_n)$$

where each X_i and Y_i, $1 \leq i \leq n$, is a variable or constant, each Z_i, $1 \leq i \leq m$, is a variable appearing in the body of the query, and each R_i, $1 \leq i \leq n$, is a regular expression over Σ.

The answer to a CRP query Q on a graph G is specified as follows. We find for each conjunct (X_i, R_i, Y_i), $1 \leq i \leq n$, a relation r_i over the scheme (X_i, Y_i) such that tuple $t \in r_i$ if there is a path from $t[X_i]$ to $t[Y_i]$ in G whose concatenation of edge labels satisfies the regular expression R_i. We then form the natural join of relations $r_1, \ldots, r_n$, $1 \leq i \leq m$, and project over the variables appearing in the head of the query.

Example 1. Suppose that Mary is studying English at university and wishes to find out what possible future career choices there are for her by seeing what other people who studied English at university have gone on to do. The following CRP query Q_0 can be formulated[1]:

[1] In the concrete syntax of CRP queries, variable names are preceded with '?'. In practice, we would expect such user querying requirements to be submitted via a

```
(?E2,?P) <- (?E1,type,University),(?E1,qualif,?D),
            (?D,type,EnglishStudies),
            (?E1,next+,?E2),
            (?E2,type,Work),(?E2,job,?A),(?A,type,?P)
```

When Q is evaluated on a database that includes the fragment of timeline data shown in Figure 1, the results returned include
`(ep22,AirTravelAssistant)`
`(ep23,Journalist)`
`(ep24,AssistantEditor).` □

We see that amongst the answers returned is a tuple showing that someone went on to become an Air Travel Assistant. Mary may judge this occupation as not requiring specifically a degree in English, and she may wish to refine her query in order to find out about future career choices that are dependent on people having studied English at University.

The timeline in Figure 1 has an edge labelled `prereq` from ep23 to ep24. This is an annotation created by the the timeline's owner indicating that this person believes that undertaking an earlier episode was necessary in order for them to be able to proceed to or achieve a later episode. So Mary might instead pose this query, Q_1:

```
(?E2,?P) <- (?E1,type,University),(?E1,qualif,?D),
            (?D,type,EnglishStudies),
            (?E1,prereq+,?E2),
            (?E2,type,Work),(?E2,job,?A),(?A,type,?P)
```

However, this will return no results relating to the example timeline of Figure 1, even though it is evident that this timeline does contain information that would be relevant to Mary. This is because, in practice, users may or may not create `prereq` metadata relating to their timelines. So it is desirable to provide users with flexible ways of querying such timeline data.

Using regular expressions to query data has been much studied (e.g. [4, 2, 17, 20]), as have approximate query matching techniques (e.g. [7, 9–11, 14–16]. In [12], we studied a combination of these and showed that approximate matching of CRP queries can be undertaken in polynomial time, building on techniques from [9]. The edit operations we allowed in approximate matching of queries were insertions, deletions and substitutions of edge labels, inversions of edge labels (corresponding to reverse traversals of edges in the database graph), and transpositions of adjacent labels, each with an assumed edit cost of 1. Query results were returned incrementally to the user in order of their increasing edit distance from the original query.

Here, we assume the same edit operations except that, for simplicity of exposition, we exclude inversions of edge labels — we note though that the techniques we describe extend straightforwardly to this more general case, and the query complexity results still hold.

---

visual search interface (similarly to the current L4All Graphical User Interface) and results to be presented in a visually appealing and easy-to-browse fashion.

Example 2. Consider query Q_1 above. If Mary chooses to allow replacement of the edge label `prereq` in her query by the label `next`, she can submit a variant of Q_1:

```
(?E2,?P) <- (?E1,type,University),(?E1,qualif,?D),
            (?D,type,EnglishStudies),
            APPROX (?E1,prereq+,?E2),
            (?E2,type,Work),(?E2,job,?A),(?A,type,?P)
```

The regular expression `prereq+` in the above query can be approximated by the regular expressions `next.prereq*` and `prereq.next.prereq*`, both at edit distance 1 from `prereq+`. This allows the system to return
`(ep22,AirTravelAssistant)`
to Mary, at an edit distance 1 from her original query Q_1. Mary may judge this result to be not relevant and may seek further results from the system, at a further level of approximation. The regular expressions `next.prereq*` and `prereq.next.prereq*` can both be approximated by `next.next.prereq*`, now at edit distance 2 from Q_1. This allows the following answers to be returned:
`(ep23,Journalist)`
`(ep24,AssistantEditor)`
Mary may judge both of these as being relevant, and she can then request the system to return the whole of this user's timeline for her to explore further. □

The previous example took as input a starting timeline episode and explored the possible future work choices. This next example additionally specifies an end goal and explores how someone might reach this from a given starting point:

Example 3. Suppose Mary knows she wants to become an Assistant Editor and would like to find out how she might achieve this, given that she's done an English degree. Mary might pose this query, Q_2:

```
(?E2,?P) <- (?E1,type,University),(?E1,qualif,?D),
            (?D,type,EnglishStudies),
            APPROX (?E1,prereq+,?E2),
            (?E2,job,?A),(?A,type,?P)
            APPROX (?E2,prereq+,?Goal),
            (?Goal,type,Work),(?Goal,job,?AG),
            (?AG,type,AssistantEditor)
```

At distance 0 there are no results from the example timeline of Figure 1. At distance 1 there are still no results. At distance 2, the following answers are returned:
`(ep22,AirTravelAssistant)`
`(ep23,Journalist)`
the second of which gives Mary potentially useful information about how she might become an Assistant Editor. □

Example 4. Continuing with Example 3, suppose Mary now wants to know what other jobs, similar to being an Assistant Editor, might be open to her. There

are many categories of jobs classified under the more general category `Media Professional`. However, if Mary enters query Q_2 above none of these jobs related to Assistant Editor will be matched by her query. What she would like to pose instead (borrowing the "RELAX" syntax from [11]) is:

```
(?E2,?P) <- (?E1,type,University),(?E1,qualif,?D),
            (?D,type,EnglishStudies),
            APPROX (?E1,prereq+,?E2),
            (?E2,job,?A),(?A,type,?P)
            APPROX (?E2,prereq+,?Goal),
            (?Goal,type,Work),(?Goal,job,?AG),
            RELAX (?AG,type,AssistantEditor)
```

which would relax the concept `Assistant Editor` to its parent concept `Editor`, matching jobs such as `Assistant Editor`, `Associate Editor`, `Editor-in-Chief` etc. Mary could seek further relaxation of `Editor` to `Media Professional`, which could return even more results. □

Example 5. As a further extension, suppose another user, Joe, wants to know what jobs similar being an Assistant Editor might be open to someone who has studied English or a similar subject at university. Subject disciplines are again classified, e.g. English Studies under `Languages`, which itself is classified under `Humanities`. So Joe may pose the query

```
(?E2,?P) <- (?E1,type,University),(?E1,qualif,?D),
            RELAX (?D,type,EnglishStudies),
            APPROX (?E1,prereq+,?E2),
            (?E2,job,?A),(?A,type,?P)
            APPROX (?E2,prereq+,?Goal),
            (?Goal,type,Work),(?Goal,job,?AG),
            RELAX (?AG,type,AssistantEditor)
```

This would successively relax both the concept `EnglishStudies` and the concept `Assistant Editor`, as well as in parallel approximating the two instances of the regular expression `prereq+`. Query results would be returned in increasing overall distance (relaxation and approximation) from the original query. □

We explored query relaxation techniques for RDF/S queries in [11], but did not consider combining this with query approximation. Here, we consider combining query approximation with a restricted form of query relaxation, in order to support the potentially complex querying requirements of lifelong learners who are searching over heterogeneous timeline-related data. Specifically, given a query conjunct of the form $(X, type, c)$, we wish to allow it to be rewritten to $(X, type, c^{sup})$, with some cost $relax(c, c^{sup})$, where c^{sup} is a superclass of c.

3 Single-Conjunct ApproxRelax Queries

In this section we introduce the notion of *ApproxRelax queries* comprising only a single conjunct. In the next section, we define *ApproxRelax queries* in general, of which the last two queries above are examples.

3.1 Single-conjunct Queries

We recall that our data model is that of a directed graph $G = (V, E)$, where each node in V is labelled with a constant and each edge e is labelled with a symbol l drawn from a finite alphabet Σ.

A *single-conjunct query* consists of an expression of the form:

$$Z_1, Z_2 \leftarrow (X, R, Y)$$

where X and Y are constants or variables, R is a regular expression, and each of Z_1 and Z_2 is one of X or Y.

A *regular expression* R over Σ is defined as follows:

$$R := \epsilon \mid a \mid \_ \mid (R1 \cdot R2) \mid (R1|R2) \mid R^* \mid R^+$$

where ϵ is the empty string, a is any symbol in Σ, "_" denotes the disjunction of all constants in Σ, and the operators have their usual meaning.

A *path* p in $G = (V, E)$ from $x \in V$ to $y \in V$ is a sequence of the form $(v_1, l_1, v_2, l_2, v_3, \ldots, v_n, l_n, v_{n+1})$, where $n \geq 0$, $v_1 = x$, $v_{n+1} = y$ and, for each $1 \leq i \leq n$, $v_i \xrightarrow{l_i} v_{i+1} \in E$. The path p *conforms* to a regular expression R if $l_1 \cdots l_n \in L(R)$, the language denoted by R.

Given a single-conjunct query Q and graph G, let θ be a matching from variables and constants of Q to nodes of G, that maps each constant to itself. A tuple $\theta(Z_1, Z_2)$ *satisfies* Q on G if there is a path from $\theta(X)$ to $\theta(Y)$ which conforms to R. The *answer* of Q on G is the set of tuples which satisfy Q on G.

3.2 Approximated single-conjunct queries

An *approximated single-conjunct query* consists of an expression of the form:

$$Z_1, Z_2 \leftarrow APPROX(X, R, Y)$$

where X, Y, R, Z_1, Z_2 are as in Section 3.1

The edit distance from a path p in G to a path p' in G is the minimum cost of any sequence of edit operations which transforms the sequence of edge labels of p to the sequence of edge labels of p' (for simplicity, in this paper we assume that all edit operations have a cost of 1). The edit distance of a path p to a regular expression R, denoted *edist*(p, R), is the minimum edit distance from p to any path that conforms to R.

Given a matching θ from variables and constants of a query Q to nodes in a graph G, the tuple $\theta(Z_1, Z_2)$ has edit distance *edist*(p, R) to Q, where p is a path from $\theta(X)$ to $\theta(Y)$ which has the minimum edit distance to R of any path from $\theta(X)$ to $\theta(Y)$ in G. Note that if p conforms to R, then $\theta(Z_1, Z_2)$ has edit distance zero to Q.

We also note that the maximum edit distance of any simple path in G to R is $|R| + |E|$ — this is because any simple path p can be accepted by an automaton

obtained from R by transitions labelled with ϵ that delete all the symbols that appear in R, followed by transitions that add all the labels of p.

The *approximate top-k answer* of Q on G is a list containing the k tuples $\theta(Z_1, Z_2)$ with minimum edit distance to Q, ranked in order of increasing edit distance.

We refer the reader to [12] for details of how the approximate top-k answer can be computed in polynomial time in the size of R and G. In brief, this is done by (i) constructing an approximate automaton, M, that recognises all paths at edit distance up to $|R| + |E|$ from R; (ii) forming the product automaton, H, of M with the database graph G; and (iii) traversing H until the first k answers have been produced. This evaluation can also be accomplished "on-demand", by incrementally constructing the edges of H as required, thus avoiding precomputation and materialisation of the entire graph H — we refer the reader to [12] for details.

3.3 Relaxed single-conjunct queries

A *relaxed single-conjunct query* consists of an expression of the form:

$$X \leftarrow RELAX(X, type, c)$$

where X is a constant or a variable, and c is a constant.

Assuming that the subgraph of G induced by edges labelled `sc` is a forest of trees, the *relaxation trail* from a triple $(X, type, c_0)$ is the sequence triples

$$(X, type, c_0), (X, type, c_1), \ldots, (X, type, c_n)$$

such that (i) for every $i \geq 0$, c_{i+1} is a superclass of c_i and there is no other class c' that is distinct from c_i and c_{i+1}, and that is a superclass of c_i and a subclass of c_{i+1}; and (ii) c_n has no superclasses.

Given a matching θ of X to nodes in G, the value $\theta(X)$ has relaxation distance 0 from c_0 if it can be inferred that $\theta(X)$ has type c_0, or relaxation distance i from c_0, for some $0 < i \leq n$ if it can be inferred that $\theta(X)$ has type c_i and does not have type c_{i-1}. We denote by $rdist(\theta(X), c_0)$ this relaxation distance.

The *relaxed top-k answer* of $(X, type, c)$ on G is a list containing the k values $\theta(X)$ with minimum relaxation distance to c, ranked in order of increasing relaxation distance to c. Note that if X is a constant, then there can be at most one answer.

To compute the relaxed top-k answer we first compute the reflexive, transitive reduction of the subgraph of G that is induced by edges labelled `sc`. This can be done "off-line" every time that this subgraph is updated by the insertion or deletion of an edge labelled `sc` (at a cost of $O(|V|^3)$ [1]) and hence need not be considered as part of query answering. We also assume that G contains all edges labelled `type` that can be inferred. Again we assume that this is done "off-line" every time that a new edge labelled `type` or `sc` is inserted or deleted (at a cost of $O(|V|^3)$).

The relaxed top-k answer of $(X, type, c_0)$ can then be computed by traversing the relaxation trail from $(X, type, c_0)$: $(X, type, c_0), (X, type, c_1), \ldots, (X, type, c_n)$, for some $n \geq 0$. During this traversal, we return all edges in G matching $(X, type, c_0)$; we then return all edges matching $(X, type, c_1)$ but not $(X, type, c_0)$; $\ldots$; all edges matching $(X, type, c_i)$ but not $(X, type, c_j)$ for $j < i$; and so forth, until either k answers have been returned, or the relaxation trail for $(X, type, c_0)$ has been completely traversed. This traversal and computation of matching edges can be accomplished in time $O(k \cdot |E|^2)$.

4 General ApproxRelax Queries

An *ApproxRelax* query Q is of the form

$$(Z_1, \ldots, Z_m) \leftarrow (X_1, R_1, Y_1), \ldots, (X_j, R_j, Y_j),$$
$$APPROX(X_{j+1}, R_{j+1}, Y_{j+1}), \ldots, APPROX(X_{j+k}, R_{j+k}, Y_{j+k}),$$
$$RELAX(X_{j+k+1}, type, c_1), \ldots, RELAX(X_{j+k+n}, type, c_n)$$

where $j, k, n \geq 0$, $X_1, \ldots, X_{j+k+n}$ and $Y_1, \ldots, Y_{j+k}$ are constants or variables, $R_1, \ldots, R_{j+k}$ are regular expressions, $c_1, \ldots, c_n$ are constants, and each Z_i is one of $X_1, \ldots, X_{j+k+n}$ or $Y_1, \ldots, Y_{j+k}$. In the concrete query syntax, the user may specify the conjuncts in the body of an ApproxRelax query in any order, and their ordering is immaterial to the query results.

Given a matching θ from variables and constants of Q to nodes in a graph G, the tuple $\theta(Z_1, \ldots, Z_m)$ has distance

$$\alpha(edist(p_1, R_{j+1}) + \cdots + edist(p_k, R_{j+k})) +$$
$$\rho(rdist(\theta(X_{j+k+1}), c_1) + \cdots + rdist(\theta(X_{j+k+n}), c_n))$$

to Q, where each p_i is the path from $\theta(X_{j+i})$ to $\theta(Y_{j+i})$ which has the minimum edit distance to R_{j+i} of any path from $\theta(X_{j+i})$ to $\theta(Y_{j+i})$ in G, and the coefficients α and ρ are set according to the preferences of the user. For example, α and ρ can be set to the same value if the same "cost" is associated with query approximation and query relaxation, or to different relative values to penalise one or the other more.

The *top-k answer* of Q on G is the list of k tuples of the form $\theta(Z_1, \ldots, Z_m)$ with minimum distance to Q, ranked in order of increasing distance to Q.

To ensure polynomial-time evaluation, we require that the conjuncts of a query Q are *acyclic* [8] (the general problem of deciding if the natural join of n relations is nonempty is NP-complete). This means that there is a join tree T whose nodes are the conjuncts of Q such that, for every variable in Q, the subgraph of T induced by the conjuncts containing the variable is connected.

Query evaluation of ApproxRelax queries is based on the hash ripple join algorithm of Ilyas et al. [13]. For each query conjunct of the form (i) (X_i, R_i, Y_i) or (ii) $(X_i, type, c_i)$, we can use the techniques described in Sections 3.2 and 3.3 to incrementally compute a relation r_i. In case (i), r_i has scheme (X_i, Y_i, ED, RD) where, for any tuple $t \in r_i$, $t[RD] = 0$ and $t[ED]$ is the minimum edit distance from R_i of any path from $t[X_i]$ to $t[Y_i]$ in G. In case (ii), r_i has scheme (X_i, ED, RD) where, for any tuple $t \in r_i$, $t[ED] = 0$ and $t[RD]$ is the relaxation distance of $t[X_i]$ to c_i.

Because the query Q is acyclic, we can construct a query evaluation tree E for Q, which consists of nodes denoting join operators and nodes representing conjuncts of Q. We initialise two hash tables for each node JN that denotes a join operator. These tables will hold the results being incrementally computed by the two children notes of the join operator, LN and RN. A "threshold" value, T_{JN}, for each node JN is initially set to 0.

Incremental evaluation of query Q proceeds by calling a recursive procedure $getNext$ starting with the root of the join tree. There are two versions of $getNext$, one for when the evaluation tree node represents a join, shown below and adapted from [13], and one for when it represents a single query conjunct, as described in Sections 3.2 and 3.3.

In procedure $getNext$ below, Q_{JN} is a priority queue associated with each join node JN, into which result tuples are placed in order of increasing overall distance value. The attribute ED of a result tuple u resulting from the join of two tuples s and t holds the combined edit distance value of s and t. Similarly, attribute RD of u holds the combined relaxation distance value of s and t. The overall distance value of u is given by $\alpha(t[ED]) + \rho(t[RD])$.

Procedure `getNext(`JN`)`

Input: node JN (corresponding to a join) in the query evaluation tree, with children LN and RN

while $empty(Q_{JN})$ **or** $(\alpha(head(Q_{JN})[ED]) + \rho(head(Q_{JN})[RD])) > T_{JN}$ **do**
- determine next input IN // either LN or RN
- $t \leftarrow getNext(IN)$
- **if** *this is the first tuple retrieved from* IN **then**
  - $IN_{top} \leftarrow \alpha(t[ED]) + \rho(t[RD])$
- $IN_{bottom} \leftarrow \alpha(t[ED]) + \rho(t[RD])$
- $T_{JN} \leftarrow \min(LN_{top} + RN_{bottom}, LN_{bottom} + RN_{top})$
- insert t into the hash table for IN
- probe the other hash table with t
- **foreach** *valid join combination* u *of* t *with* s, *say,* **do**
  - $u[ED] \leftarrow t[ED] + s[ED]$
  - $u[RD] \leftarrow t[RD] + s[RD]$
  - $enqueue(Q_{JN}, u)$

return $dequeue(Q_{JN})$

We see that the $getNext$ procedure for a join node JN begins by choosing from which of its two operands, LN or RN, to retrieve tuples. There are various heuristics that might be used to decide this — see [13]. For input IN (either LN or RN), IN_{top} represents the distance value of the first tuple retrieved from IN (i.e., the smallest distance in IN), while IN_{bottom} represents the distance of the most recently retrieved tuple from IN. The threshold value T_{JN} represents the smaller of the two distances given by $LN_{top} + RN_{bottom}$ and $LN_{bottom} + RN_{top}$. These two values give the possible distances arising from joining the first

tuple of LN with the most recent tuple from RN, or vice versa, either of which remains possible until the end of the $getNext$ operation. The smaller of these two distances gives the smallest possible distance for a join tuple that has yet to be computed; in other words, no tuple that might result from the join of tuples yet to be retrieved with tuples already retrieved or yet to be retrieved can have a distance less than T_{JN}. It is therefore safe to output a join tuple whose distance is equal to T_{JN}.

5 Concluding Remarks

Facilitating the collaborative formulation of learning goals and career aspirations has the potential to enhance learners' engagement with the lifelong learning process. The L4All system offers similarity matching over learners' timelines in order to identify possible choices for future learning and professional development. Here, we have extended this idea by combining query approximation and query relaxation techniques, in order to provide greater flexibility in querying of heterogeneous timeline data. Using the techniques proposed here, users would be able to specify approximations and relaxations to be applied to their original search query, and the relative costs of these. Query results would be returned incrementally by the system, ranked in order of increasing "distance" from the user's original query. We have presented a polynomial-time algorithim for incrementally computing the top-k answers to such queries.

In practice, we expect that a visual query interface would be required, providing users with readily understandable options from which to select their query formulation, approximation and relaxation requirements, and set the relative cost associated with each query rewrite operation they have selected. An open question is the degree of domain expertise that would be necessary to use effectively such an interface, leading possibly to the design of different interfaces for learners and for trained careers advisors. Our future work includes the design and prototyping of such a query interface, or interfaces, the empirical evaluation of our query processing algorithms over timeline data gathered from earlier L4All user evaluation sessions, and the evaluation of the new querying facilities with groups of lifelong learners at our institution and other partner FE and HE institutions in London. At present, the L4All system stores all users' timeline data and metadata within a single RDF/S repository, and another direction of future work would be to extend our query processing techniques to query *linked data* repositories of timeline information.

References

1. A. Aho, M. Garey, and J. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1(2):131–137, 1972.
2. D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. Seventh Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 176–185, 2000.

3. C. Cogoi. Using ICT in Guidance: Practitioner Competencies and Training. *Report of an EC Leonardo project on ICT Skills for Guidance Counsellors. Bologna: Outline Edizone*, 2005.
4. I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. In *Proc. ACM SIGMOD Conf.*, pages 323–330, 1987.
5. L. Cych. 'Social Networks' in Emerging Technologies for Learning, Coventry. *Becta*, pages 32–40, 2006.
6. S. de Freitas, I. Harrison, G. Magoulas, A. Mee, F. Mohamad, M. Oliver, G. Papamarkos, and A. Poulovassilis. The development of a system for supporting the lifelong learner. *British Journal of Educational Technology*, 37(6):867–880, 2006.
7. P. Dolog, H. Stuckenschmidt, and H. Wache. Robust query processing for personalized information access on the semantic web. In *Proc. 7th Int. Conf. on Flexible Query Answering Systems*, pages 343–355, 2006.
8. G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 43(3):431–498, May 2001.
9. G. Grahne and A. Thomo. Approximate reasoning in semi-structured databases. In *Proc. 8th Int. Workshop on Knowledge Representation meets Databases*, 2001.
10. G. Grahne, A. Thomo, and W. W. Wadge. Preferentially annotated regular path queries. In *Proc. 11th Int. Conf. on Database Theory*, pages 314–328, 2007.
11. C. A. Hurtado, A. Poulovassilis, and P. T. Wood. Query relaxation in RDF. *Journal on Data Semantics*, X:31–61, 2008.
12. C. A. Hurtado, A. Poulovassilis, and P. T. Wood. Ranking approximate answers to semantic web queries. In *Proc. 6th European Semantic Web Conference*, pages 263–277, 2009.
13. I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. *The VLDB Journal*, 13:207–221, 2004.
14. H. V. Jagadish, A. O. Mendelzon, and T. Milo. Similarity-based queries. In *Proc. Fourteenth ACM Symp. on Principles of Databases Systems*, pages 36–45, 1995.
15. Y. Kanza and Y. Sagiv. Flexible queries over semistructured data. In *Proc. Twentieth ACM Symp. on Principles of Databases Systems*, pages 40–51, 2001.
16. C. Kiefer, A. Bernstein, and M. Stocker. The fundamentals of iSPARQL: A virtual triple approach for similarity-based semantic web tasks. In *Proc. 6th Int. Semantic Web Conf.*, pages 295–309, 2007.
17. K. Kochut and M. Janik. SPARQLeR: Extended SPARQL for semantic association discovery. In *Proc. 4th European Semantic Web Conference*, pages 145–159, 2007.
18. R. Koper, B. Giesbers, P. van Rosmalen, P. Sloep, J. van Bruggen, C. Tattersall, H. Vogten, and F. Brouns. A design model for lifelong learning networks. *Interactive Learning Environments*, 13(1–2):71–92, 2005.
19. R. Koper and C. Tattersall. New directions for lifelong learning using network technologies. *British Journal of Educational Technology*, 35(6):689–700, 2004.
20. J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. In *Proc. 7th Int. Semantic Web Conf.*, pages 66–81, 2008.
21. N. van Labeke, G. D. Magoulas, and A. Poulovassilis. Searching for "people like me" in a lifelong learning system. In *Proc. 4th European Conf. on Technology Enhanced Learning (EC-TEL'09) (to appear)*.
22. N. van Labeke, A. Poulovassilis, and G. D. Magoulas. Using similarity metrics for matching lifelong learners. In *Proc. 9th Int. Conf. on Intelligent Tutoring Systems (ITS'08)*, pages 142–151, 2008.