Birkbeck
UNIVERSITY OF LONDON

Flexible querying of heterogeneous graph data

Alex Poulovassilis
Birkbeck, University of London

NLDB 2013, June 2013

londonknowledgelab

Collaborators

- Riccardo Frosini, PhD student
- Petra Selmer, PhD student
- Andrea Cali
- Peter Wood
- Carlos Hurtado

Outline of the talk

1. Motivation
2. Overview of our approach
3. Single-conjunct regular path queries
    - Approximate matching of queries
    - Query relaxation
4. Multi-conjunct queries
5. The ApproxRelax prototype
6. Combining APPROX and RELAX into FLEX
7. Extending with text similarity measures
8. Current work

1. Motivation

- Increasing volumes of semi-structured data are being made available on the web e.g. as Linked Open Data
- Complexity, heterogeneity, dynamicity of such data means that users may not be aware of its full structure
- Need to be assisted by querying systems that are not limited to exact matching of users' queries
- In recent work we have been investigating combining
  - *approximate matching* and
  - *relaxation*

  of users' queries on graph data, with query answers being returned ranked according to increasing "distance" from the original query

2. Overview of our approach

- We consider semi-structured data modelled as a graph structure, e.g. RDF linked data is one kind of data that can be represented this way
- Our data model is a directed graph G = (V,E) where
  - each node in V is labelled with a constant
  - each edge e in E is labelled with a label drawn from a finite alphabet Σ
- Much of our work has been in the context of *conjunctive regular path queries*:

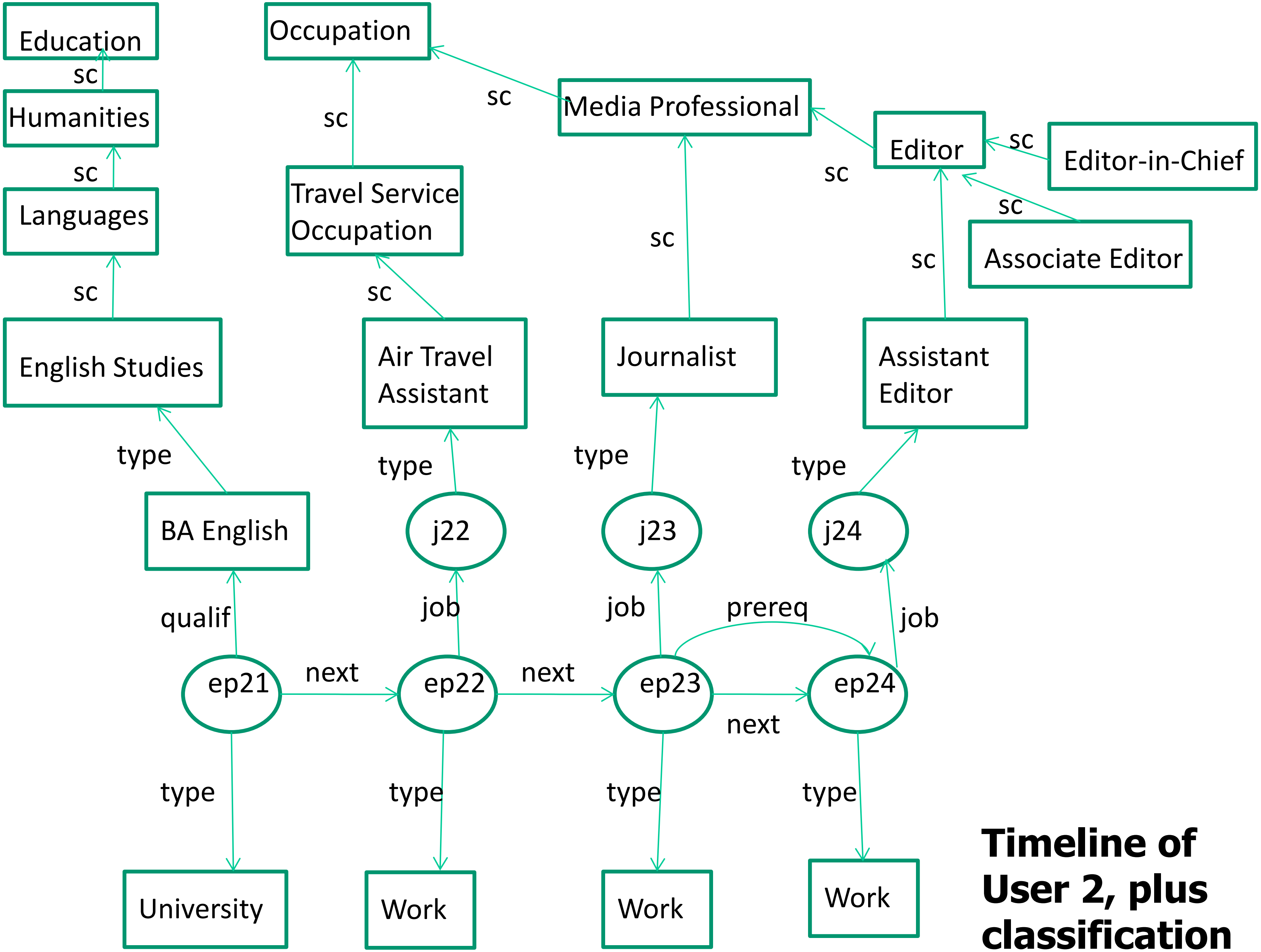
$$(Z_1, \ldots, Z_m) \leftarrow (X_1, R_1, Y_1), \ldots, (X_n, R_n, Y_n)$$

where the X_i , Y_i are variables or constants;

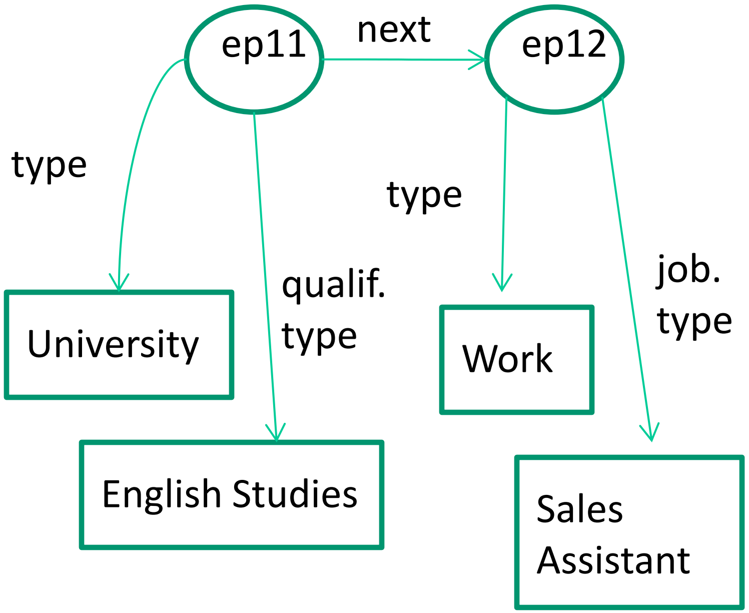
the R_i are *regular expressions* over the alphabet of edge labels, Σ; and the Z_i are variables that also appear in the query body

Example – Querying lifelong learner metadata

- The L4All system developed in JISC-funded research at the London Knowledge Lab allows users to maintain a chronological record of their episodes of learning and work: their personal "timelines"
- Users can search over the timeline data of others, and identify possible choices for their own future learning and professional development by seeing what others with a similar background have gone on to do
- However the L4All search facility is rather limited:
  - it offers a fixed set of similarity metrics over the timeline data
  - applied to just one level of detail of the classifications of selected categories of episode in the search query
- We have investigated how query approximation and relaxation techniques can be used to provide a more flexible search facility in this application domain

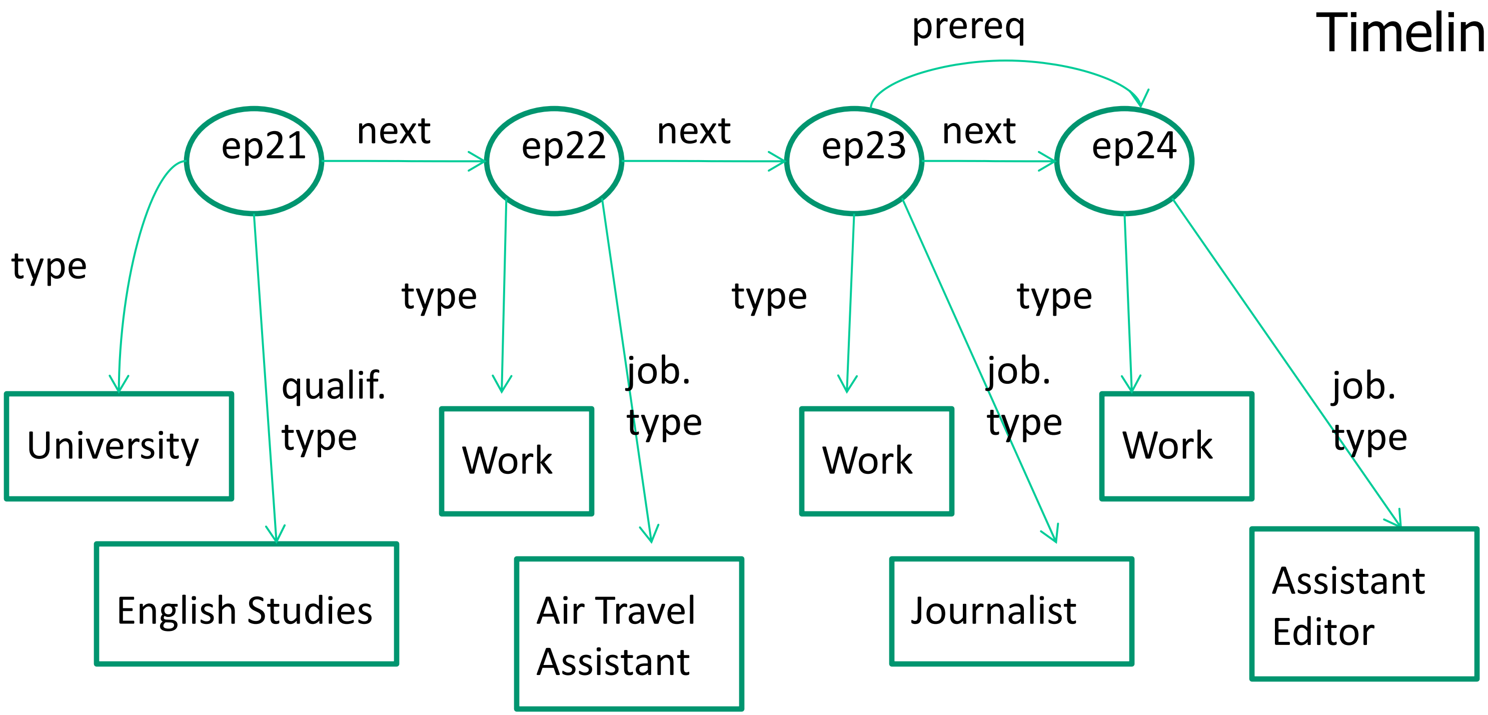


Timeline of User 2, plus classification information



A fragment of **My timeline** data and metadata

Timeline of User 2

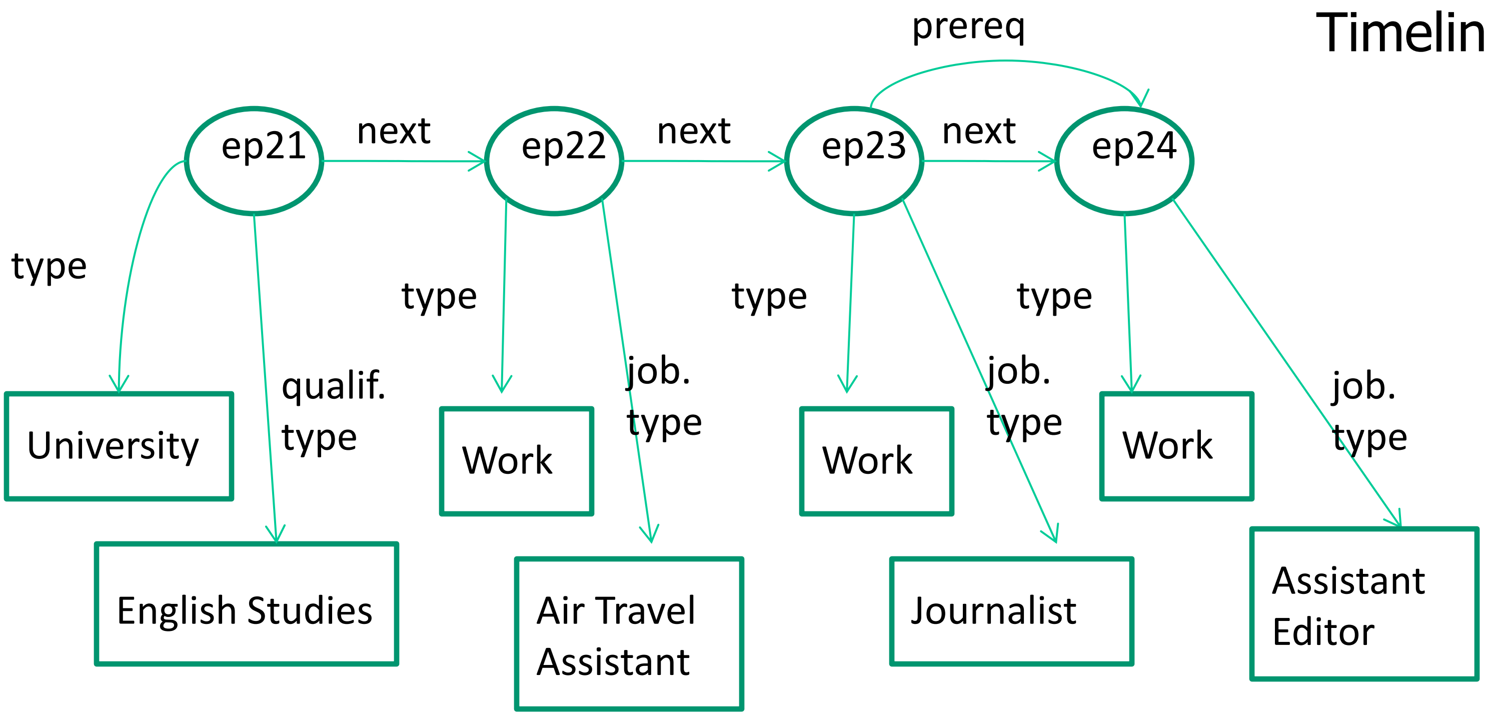


Query 1: I've done this so far. What work positions can I reach and how? E.g. selecting just the relevant prefix of my timeline (my English degree, rather than my temporary work as a Sales Assistant):

(?E2,?P) ← (?E1,type,University),(?E1,qualif.type,EnglishStudies),
(?E1,prereq+,?E2),
(?E2,type,Work),(?E2,job.type,?P)

However, this will return no results relating to User 2

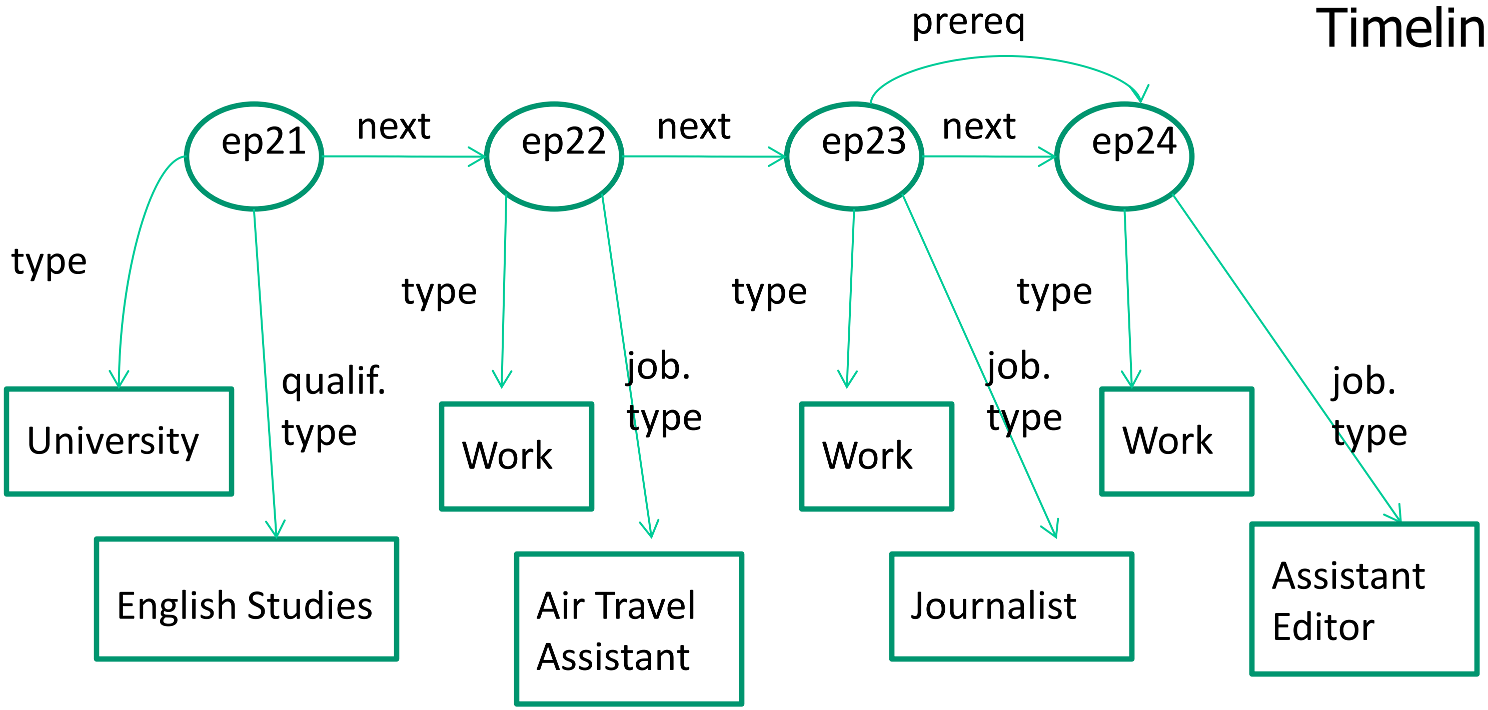
Timeline of User 2



Allowing *query approximation* can yield some answers. In particular, allowing replacement of the edge label "prereq" by the label "next", at an edit cost of 1, we can submit this variant of Query 1:

(?E2,?P) ← (?E1,type,University),(?E1,qualif.type,EnglishStudies),
APPROX (?E1,prereq+,?E2),
(?E2,type,Work),(?E2,job.type,?P)

Timeline of User 2

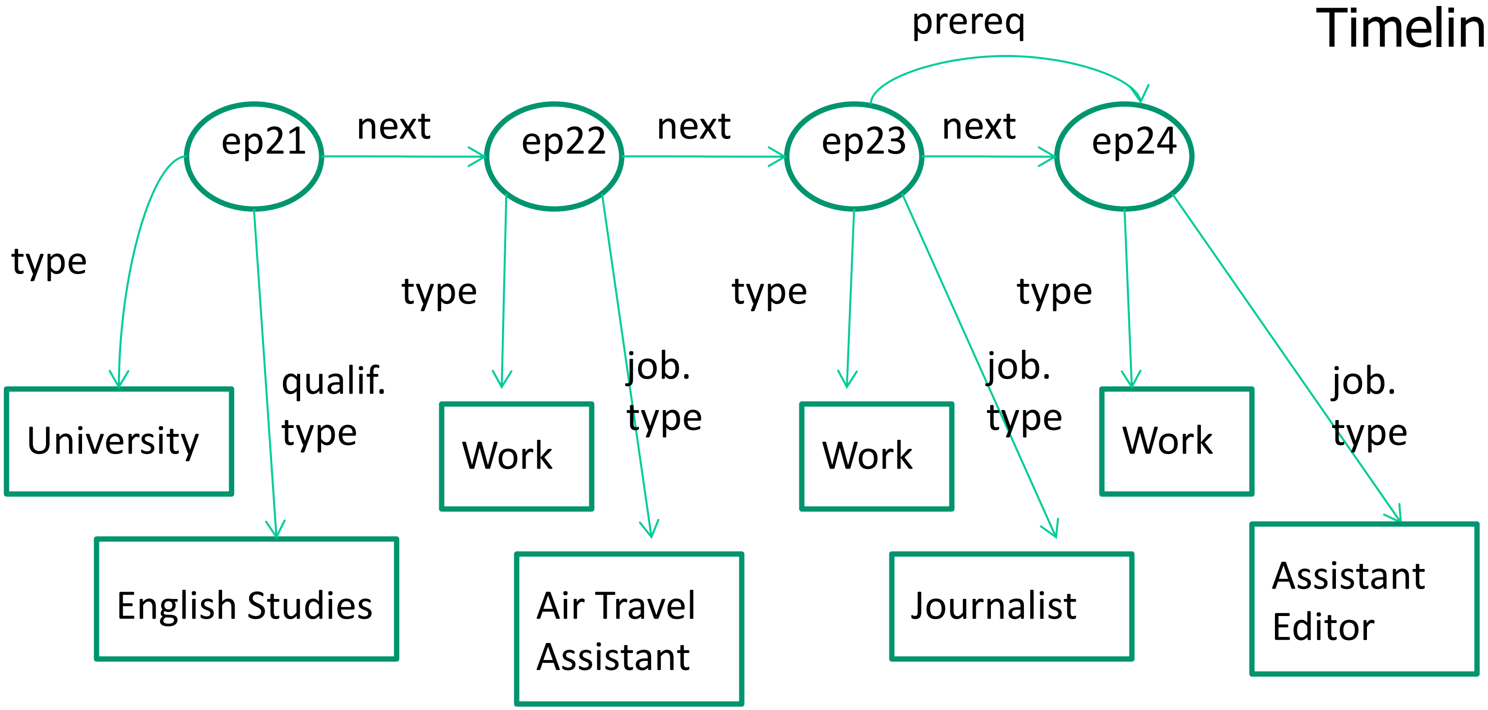


The regular expression prereq+ can be approximated by next.prereq* at edit distance 1. This allows the system to return

(ep22,AirTravelAssistant)

We may judge this result to be not relevant and seek further results from the system at a further level of approximation

Timeline of User 2

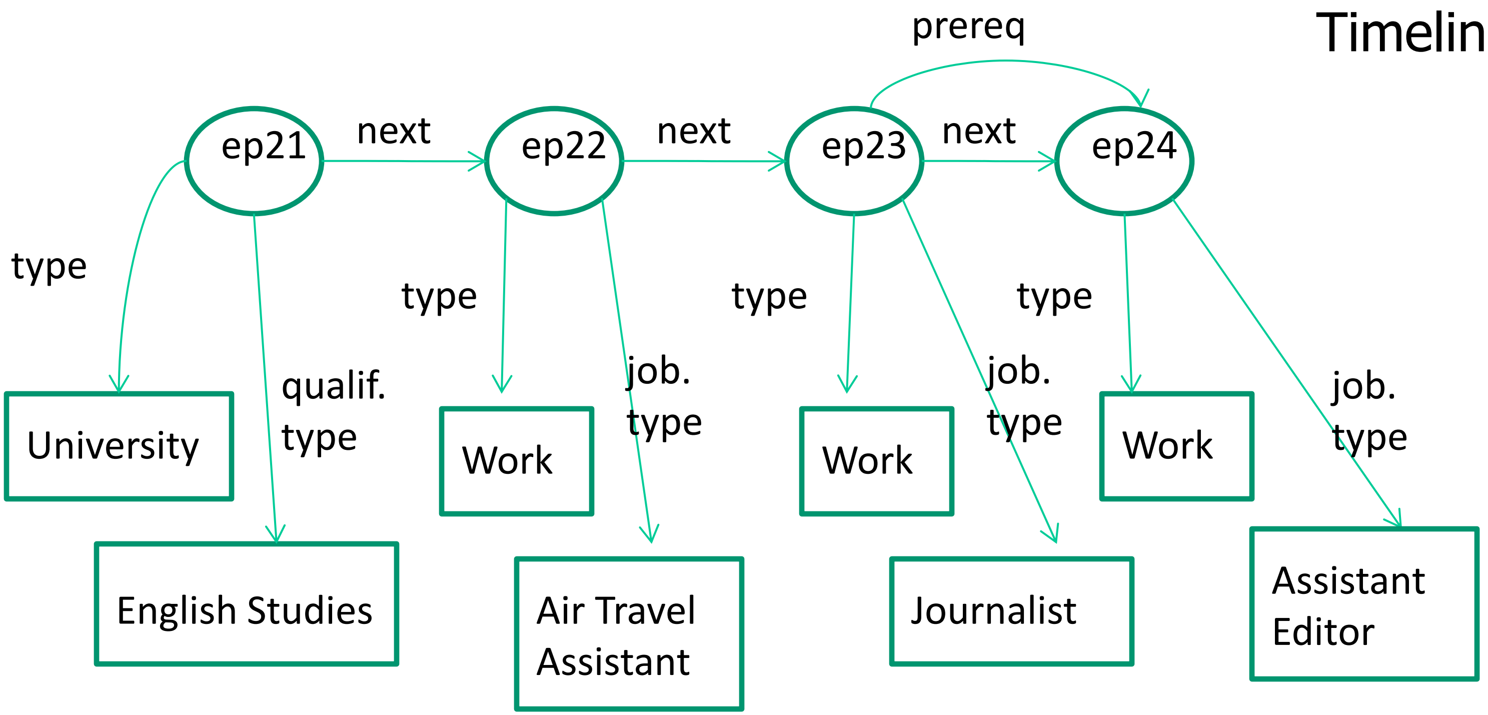


next.prereq* can be approximated by next.next.prereq*, now at edit distance 2. This allows the following answers to be returned:

(ep23,Journalist), (ep24,AssistantEditor)

We may judge both of these as being relevant, and can then request the system to return the whole of User 2’s timeline to explore further.

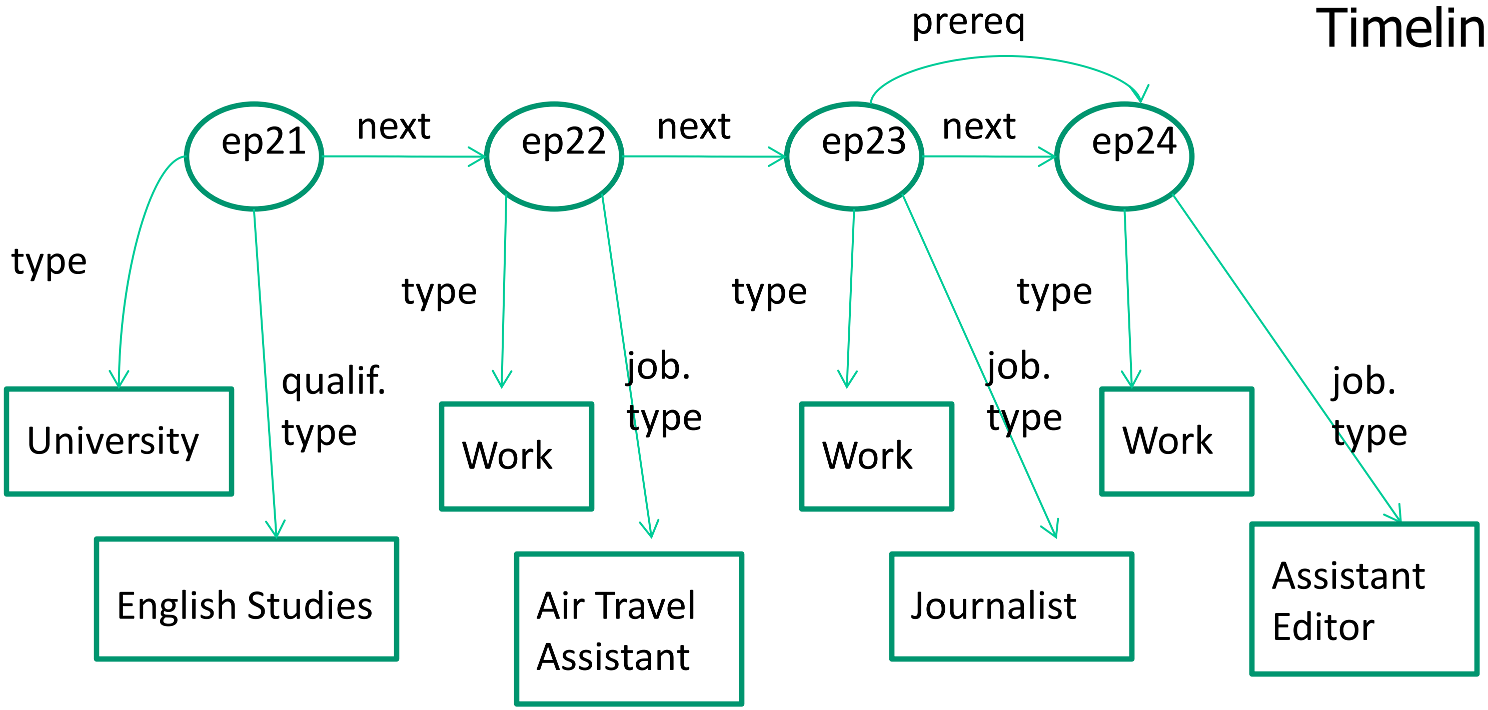
Timeline of User 2



Query 2: I want to become an Assistant Editor. How might I achieve this given that I've done an English degree:

(?E2,?P) ← (?E1,type,University),(?E1,qualif.type,EnglishStudies),
APPROX (?E1,prereq+,?E2), (?E2,job.type,?P),
APPROX (?E2,prereq+,?Goal), (?Goal,type,Work)
(?Goal,job.type,AssistantEditor)

Timeline of User 2



At distance 0 and 1 there are no results from the timeline of User 2. At distance 2, the answers

(ep22,AirTravelAssistant), (ep23,Journalist)

are returned, the second of which gives potentially useful information

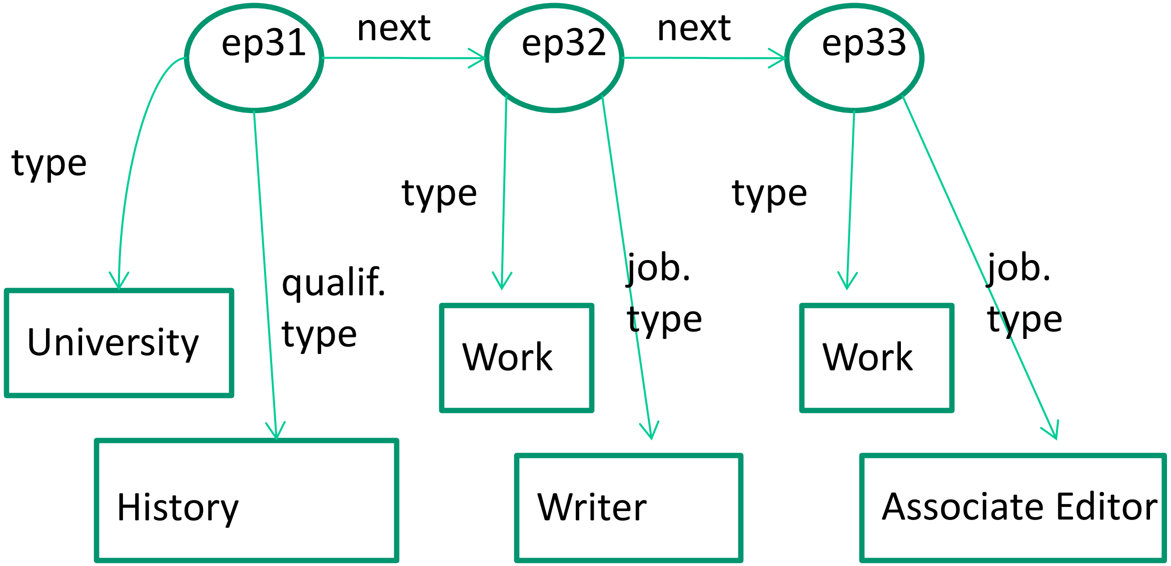
Query 3: Suppose I want to know what other jobs, similar to Assistant Editor, might be open to me.

Rather than Query 2:

```
(?E2,?P) ← (?E1,type,University),(?E1,qualif.type,EnglishStudies),
           APPROX (?E1,prereq+,?E2), (?E2,job.type,?P),
           APPROX (?E2,prereq+,?Goal), (?Goal,type,Work)
           (?Goal,job.type,AssistantEditor)
```

I can pose instead:

```
(?E2,?P) ← (?E1,type,University),(?E1,qualif.type,EnglishStudies),
           APPROX (?E1,prereq+,?E2), (?E2,job.type,?P),
           APPROX (?E2,prereq+,?Goal), (?Goal,type,Work)
           RELAX (?Goal,job.type,AssistantEditor)
```



Timeline of another user, User 3

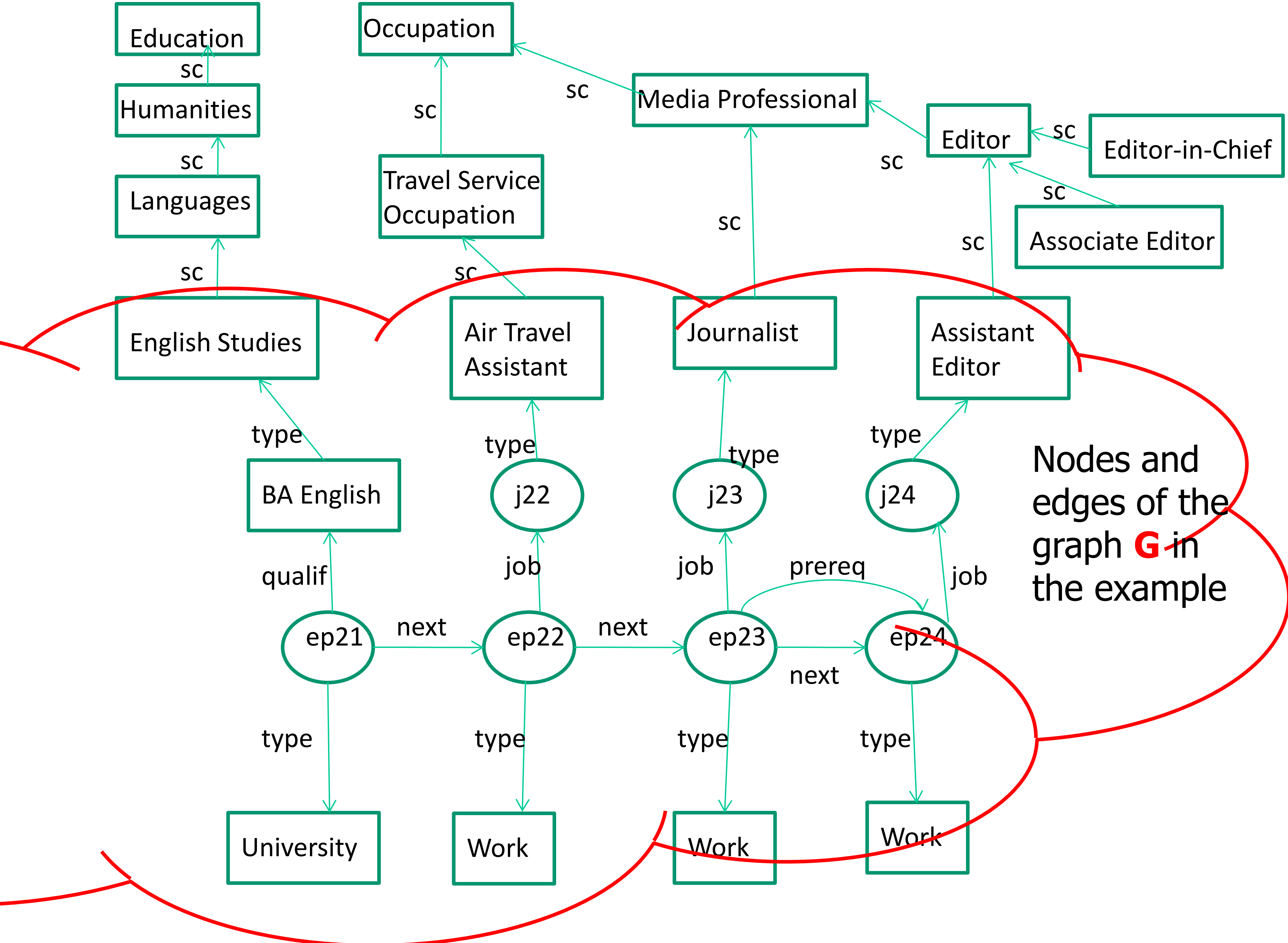
Query 4: Suppose another user, Joe, wants to know what jobs similar to Assistant Editor might be open to someone who has studied English or a similar subject at university:

(?E2,?P) ← (?E1,type,University), **RELAX**(?E1,qualif.type,EnglishStudies),
APPROX (?E1,prereq+,?E2), (?E2,job.type,?P),
APPROX (?E2,prereq+,?Goal), (?Goal,type,Work)
RELAX (?Goal,job.type,AssistantEditor)

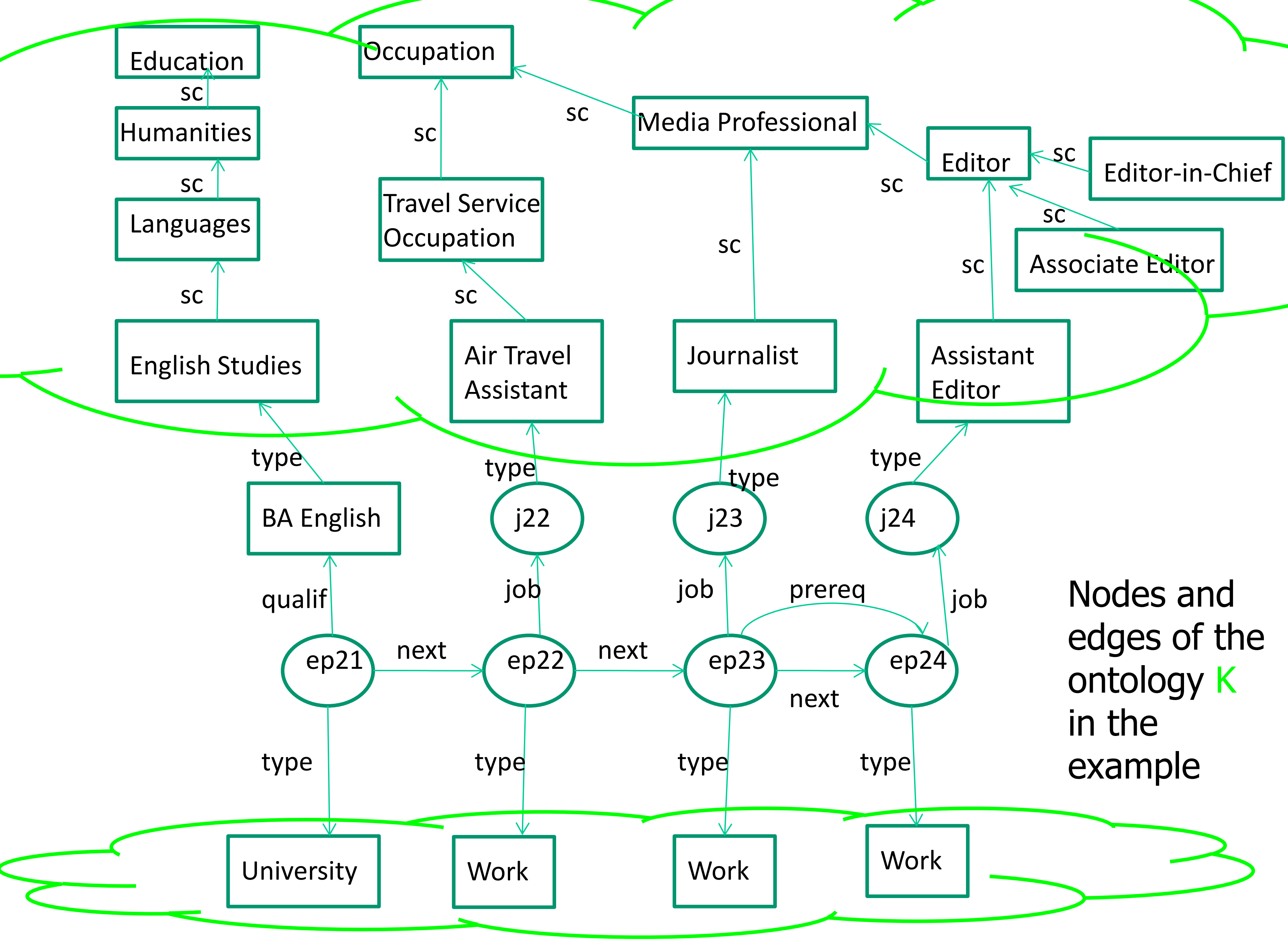
3. Single-conjunct regular path queries

- We consider a semi-structured data model comprising a directed graph $G = (V,E)$ and an ontology $K = (V_K,E_K)$
- V contains nodes representing entity instances or entity classes, each labelled with a distinct constant
- Each edge in E is labelled with a symbol drawn from a finite alphabet Σ U {type}
- V_K contains nodes representing entity classes or properties, each labelled with a distinct constant
- Each edge in E_K is labelled with a symbol drawn from

  {sc,sp,dom,range}
- This model encompasses RDF data, except for blank nodes. Plus a fragment of the RDFS vocabulary: rdf:type, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, rdfs:range



Nodes and edges of the graph **G** in the example



Nodes and edges of the ontology K in the example

Single-conjunct regular path queries

- A single-conjunct regular path query, Q, is of the form

  vars ← (X, R, Y)

  where X and Y are constants or variables, *vars* is the subset of {X,Y} that are variables, and R is a regular expression over Σ′ =Σ U {type}

- A *path* p in graph G is a sequence of the form

  $v_1 , l_1 , v_2 , l_2 , \ldots, v_n , l_n \, v_{n+1}$

  where for each v_i , v_{i+1} there is an edge $v_i \rightarrow v_{i+1}$ labelled l_i in G

- A path p *conforms to* regular expression R if $l_1 \ldots l_n$ is in the language denoted by R, *L*(R)

Exact Matching

- Given a single-conjunct regular path query Q:

  vars ← (X, R, Y)

- Let θ be a matching from {X, Y} to the nodes of graph G, that maps each constant to itself

- θ(*vars*) *satisfies* Q on G if there is a path from θ(X) to θ(Y) that conforms to R

- The *exact answer* of Q on G is the set of tuples θ(*vars*) that satisfy Q on G

Approximate Matching

- The *edit distance* from a p to a path p′ 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′

- The edit operations we consider are insertions, deletions and substitutions of edge labels, each with an assumed cost of α

  - We envisage the user/implementor being able to specify which edit operations should be applied by the system when answering a particular query, for a particular application, and the cost of each of these

Approximate Matching

- The *edit distance* of a path p to a regular expression R is the minimum edit distance from p to any path that conforms to R

- The tuple θ(*vars*) has *edit distance* *edist*(θ,Q) *to* Q, and we define this to be the minimum edit distance to R of any path from θ(X) to θ(Y) in G

- The *approximate answer of Q on G* is a list of pairs

  (θ(*vars*), *edist*(θ,Q))

  ranked in order of non-decreasing edit distance

Computing the approximate answer

1. Construct a *weighted* NFA, M_R , of size O(|R|) to recognise $\mathcal{L}$(R). The weight labels on the transitions are 0.
2. Construct the *approximate automaton* A_R corresponding to M_R. This has the same states as M_R and additional transitions, weighted α, corresponding to insertions of edge labels, deletions and substitutions. Hence A_R has $O(|R||\Sigma'|)$ transitions
3. Form the weighted *product automaton* H = A_R x G viewing each node in G as both an initial and a final state. So H has O(|R||V|) nodes and $O(|R||\Sigma'||E|)$ edges
4. If X is a node v of G, perform a *shortest path traversal* of H, starting from vertex (s_0,v) . If X is a variable, do this for every node v of G.
   - The complexity of this traversal is $O(|R|^2|V| (|\Sigma'||E| + |V| \log (|R||V|)))$ – see ISWC 2010 paper

Computing the approximate answer

- Can be accomplished “on demand” by incrementally constructing the edges of H, using a function *Succ*(s,n) which returns the set of transitions that would be the successors of vertex (s,n) in H
- Begin by enqueueing in a priority queue, $queue_R$, initial quadruples (v,v, s_0,0) for each node v in G (unless X is a specific node v, in which case only (v,v, s_0,0) is enqueued)
- Call procedure *getNext* to return the next query answer, in order of non-decreasing distance from Q:
  - *getNext* repeatedly dequeues the first quadruple (v,n,s,d) from $queue_R$ adding (v,n,s) to a set $visited_R$, until $queue_R$ is empty
  - After dequeueing (v,n,s,d) it enqueues (v,m,s’,d+d’) for each transition (s,n) → (s’,m) labelled (e,d’) that is returned by calling *Succ*(s,n) such that (v,m,s’) is not already in $visited_R$
  - If s is a final state and answer (v,n) has not been generated before, then return (v,n,d) as an answer

Query Relaxation

- We apply RDFS inference rules in order to relax regular path queries:

$$(1)\ \frac{(a, subPropertyOf, b)\ (b, subPropertyOf, c)}{(a, subPropertyOf, c)} \qquad (2)\ \frac{(a, subPropertyOf, b)\ (X, a, Y)}{(X, b, Y)}$$

$$(3)\ \frac{(a, subClassOf, b)\ (b, subClassOf, c)}{(a, subClassOf, c)} \qquad (4)\ \frac{(a, subClassOf, b)\ (X, type, a)}{(X, type, b)}$$

$$(5)\ \frac{(a, domain, c)\ (X, a, Y)}{(X, type, c)} \qquad (6)\ \frac{(a, range, c)\ (X, a, Y)}{(Y, type, c)}$$

Query Relaxation

- We consider the cost of relaxing a triple pattern (X,type,a) to (X,type,b), where b is an immediate superclass of a, to be β
    - likewise the cost of relaxing (X,a,Y) to (X,b,Y), where b is an immediate superproperty of a
- We consider the cost of relaxing (X,a,Y) to (X,type,c), where the domain of a is c, to be γ
    - likewise the cost of relaxing (X,a,Y) to (Y,type,c) , where the range of a is c

- In general, such *triple patterns* appear within a query as part of a set of triple patterns i.e. a *graph pattern*

Query Relaxation

- Using what we term the *extended reduction* of the ontology allows us to perform *direct* relaxations. These correspond to the “smallest” relaxation steps, with costs β or γ

- Given a query Q with a single conjunct (X,R,Y), let q =

  $l_1 l_2 \ \ldots \ l_n$

  be a string in $\mathcal{L}$(R). We define a ***triple form*** of (Q,q) to be a set of triples

  $\{(X, l_{1,} W_1), (W_1, l_{2,} W_2), \ldots, (W_{n-1}, l_{n,} Y)\}$

  where the W_i are variables not appearing in Q

- Thus, a triple form of (Q,q) is a graph pattern that can be relaxed to another graph pattern

Query Relaxation

- The *triple form of a path* v_1 , l_1 , v_2 , l_2 , …, v_n , l_n v_{n+1} in G is defined similarly to be a set of triple patterns

  $\{(v_1 , l_{1,} W_1), (W_1, l_{2,} W_2), \ldots, (W_{n-1}, l_{n,} v_{n+1})\}$

- Given a single-conjunct regular path query Q,

  vars ← (X, R, Y)

  let θ be a matching from variables and constants of Q to nodes of graph G, that maps each constant to itself

- We denote (θ(X),R, θ(Y)) by θ(Q). Path p in G *r-conforms* to θ(Q) if there is a string q in $\mathcal{L}$(R), a triple form T_q of (θ(Q),q) and a triple form T_p of path p such that T_q relaxes to T_p.

- A tuple θ(*vars*) *r-satisfies* Q on G if there is a path in G that r-conforms to θ(Q)

Relaxed answer

- The *relaxation distance* from path p to θ(Q) is the minimum relaxation distance from p to (θ(Q),q) for any string q in $\mathcal{L}$(R).

- The *relaxation distance* of θ(Q) from Q, denoted *rdist* (θ,Q), is the minimum relaxation distance to θ(Q) from any path p that r-conforms to R

- The *relaxed answer of Q on G* is a list of pairs

  (θ(*vars*), *rdist* (θ,Q))

  where θ(*vars*) r-satisfies Q on G, ranked in order of non-decreasing relaxation distance

Computing the relaxed answer

1. Construct a *weighted* NFA, M_R , of size O(|R|) to recognise $\mathcal{L}$(R). The weight labels on the transitions are 0.
2. Construct the *relaxed automaton* M_R^K corresponding to M_R with respect to the (extended, reduced) ontology K by adding all transitions that can be inferred from the RDFS inference rules:
   - for the cost of a new transition, add β or γ to the cost of the transition from which it was inferred

   M_R^K has at most O(|R||K|) nodes and O(|R||K|^2) transitions
3. Form the *product automaton* H = M_R^K x G viewing each node in G as both an initial and a final state. So H has O(|R||K||V|) nodes and O(|R||K|^2|E|) edges
4. If X is a node v of G, perform a *shortest path traversal* of H, starting from vertex (s_0,v) , for all initial states s_0. If X is a variable, do this for every node v of G.

Computing the relaxed answer

- Can be accomplished “on demand” by incrementally constructing the edges of H, using function *Succ*(s,n) to return the set of transitions that would be the successors of vertex (s,n) in H
- Begin by enqueueing in the priority queue initial quadruples (v,v, s_0,0) for each node v in G and initial state s_0 in M_R^K (unless X is a specific node v, in which case only (v,v, s_0,0) is enqueued)
- Call the same procedure *getNext* as used for approximate query answering to return the next query answer, in order of non-decreasing relaxation distance from Q

4. Multi-conjunct queries

- In general, a 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 $(X_{j+k+1}, R_{j+k+1}, Y_{j+k+1}), \ldots,$ RELAX $(X_{j+k+n}, R_{j+k+n}, Y_{j+k+n})$

- Given a matching θ from variables and constants of Q to nodes in graph G, the *distance* from θ to Q, *dist*(θ,Q), is sum of
  - the edit distances of conjuncts $(X_{j+1}, R_{j+1}, Y_{j+1}), \ldots, (X_{j+k}, R_{j+k}, Y_{j+k})$, and
  - the relaxation distances of conjuncts $(X_{j+k+1}, R_{j+k+1}, Y_{j+k+1}), \ldots, (X_{j+k+n}, R_{j+k+n}, Y_{j+k+n})$

Multi-conjunct queries

- θ is a *minimum distance* matching if for all matchings φ from Q to G such that $\theta(Z_1, \ldots, Z_m) = \varphi(Z_1, \ldots, Z_m)$, *dist*($\theta$,Q) ≤ *dist*($\varphi$,Q)

- The *answer of Q on G* is a list of pairs ($\theta(Z_1, \ldots, Z_m)$,*dist*(θ,Q)), for some minimum-distance matching θ, ranked in order of non-decreasing distance

Multi-conjunct query evaluation

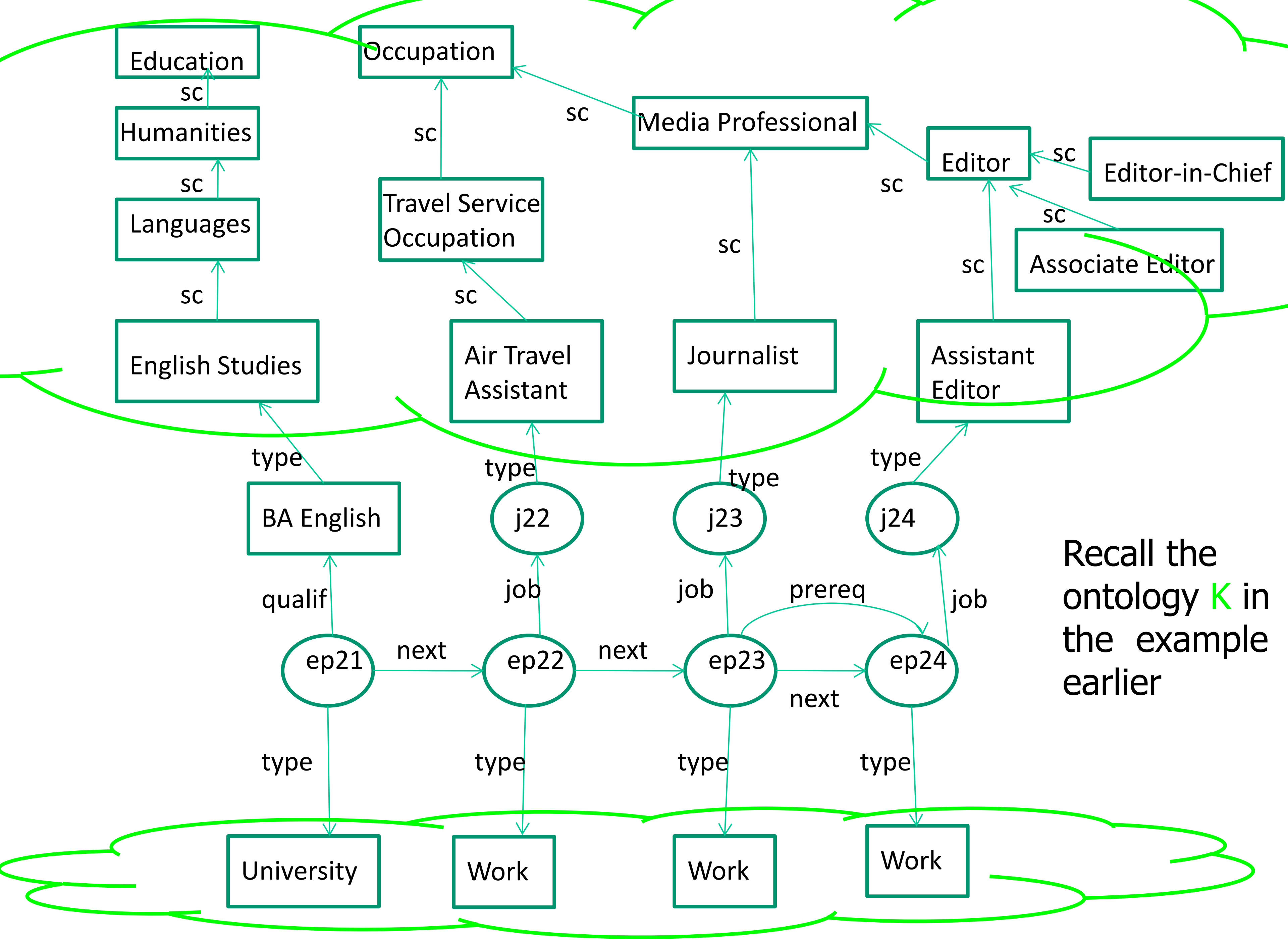
- For each query conjunct, use techniques described earlier to compute a relation r_i with scheme (X_i, Y_i, ED, RD)
- If it is an exact conjunct, then for each tuple t in r_i t[ED] = r[RD] = 0
- If it is an APPROXed conjunct, then for each tuple t in r_i t[RD] = 0 and t[ED] is the edit distance for t
- If it is a RELAXed conjunct, then for each tuple t in r_i t[ED] = 0 and t[RD] is the relaxation distance for t
- To ensure polynomial time evaluation, the conjuncts must be acyclic
- A query evaluation tree for Q can be constructed, consisting of nodes denoting join operators and nodes denoting conjuncts
- Use pipelined execution of any rank-join operator to produce answers to Q in order of non-decreasing distance

Example

- Recall **Query 4:** Joe, wants to know what jobs similar to Assistant Editor might be open to someone who has studied English or a similar subject at university:

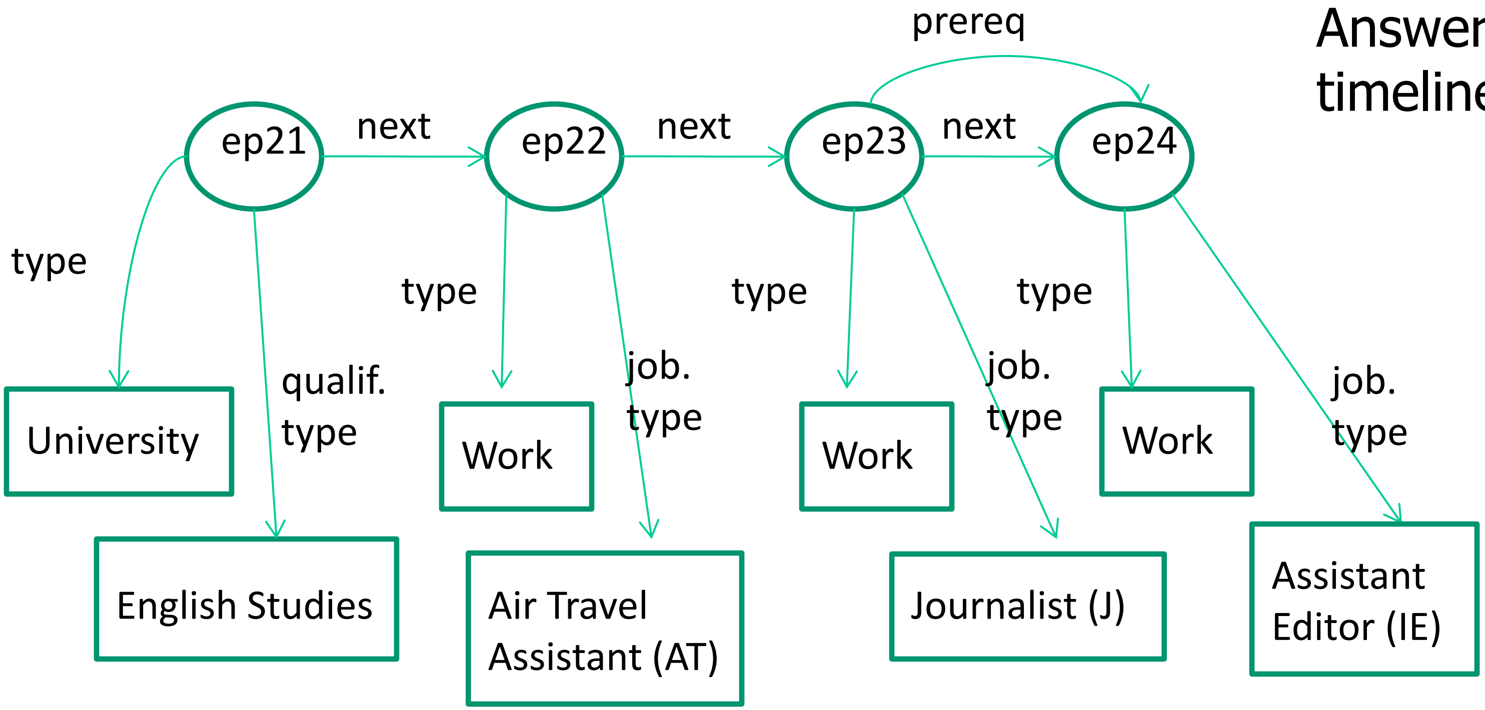
  (?E2,?P) ← (?E1,type,University),
  RELAX(?E1,qualif.type,EnglishStudies),
  APPROX (?E1,prereq+,?E2), (?E2,job.type,?P),
  APPROX (?E2,prereq+,?Goal), (?Goal,type,Work)
  RELAX (?Goal,job.type,AssistantEditor)

- Suppose α is set to 1, and β and γ are set to 2



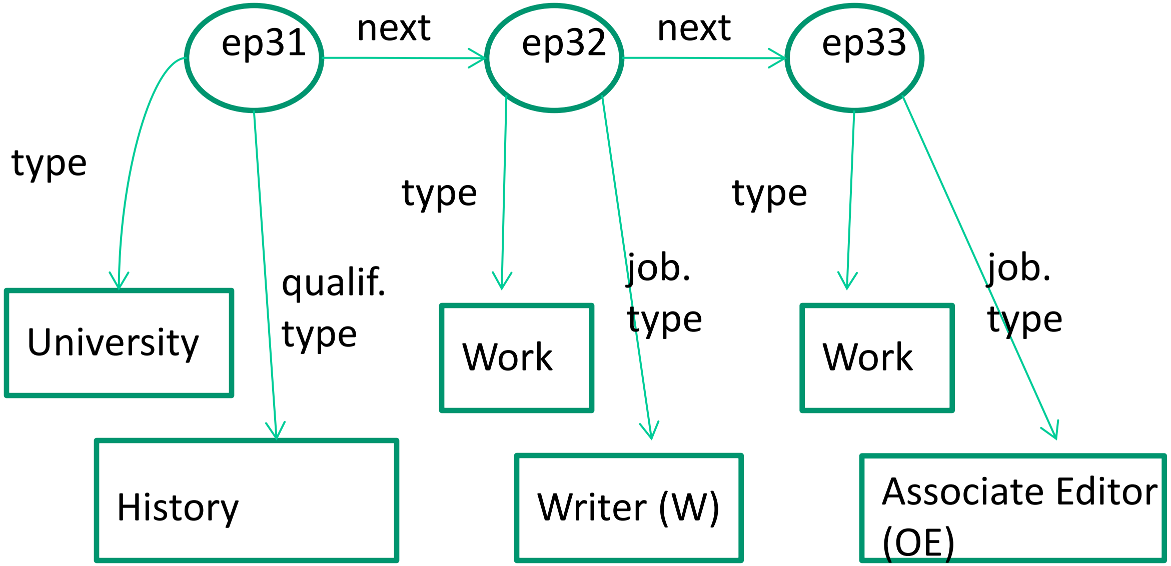
Recall the ontology K in the example earlier

Answers from timeline of User 2



?E1	**?E1,RD**	**?E1,?E2,ED**	**?E2,?P**	**?E2,?Goal,ED**	**?Goal**	**?Goal,RD**	**?E2,P,D**
ep21	*ep21,0*	ep23,ep24,0	*ep22,AT*	*ep23,ep24,0*	ep22	*ep24,0*	*ep23,J,2*
		ep21,ep22,1	*ep23,J*	ep21,ep22,1	*ep23*	*ep23,4*	*ep22,AT,6*
		ep22,ep23,1	ep24,IE	*ep22,ep23,1*	*ep24*	ep22,6	
		ep21,ep23,2		ep21,ep23,2			
		ep21,ep24,2		ep21,ep24,2			

Answers from timeline of User 3

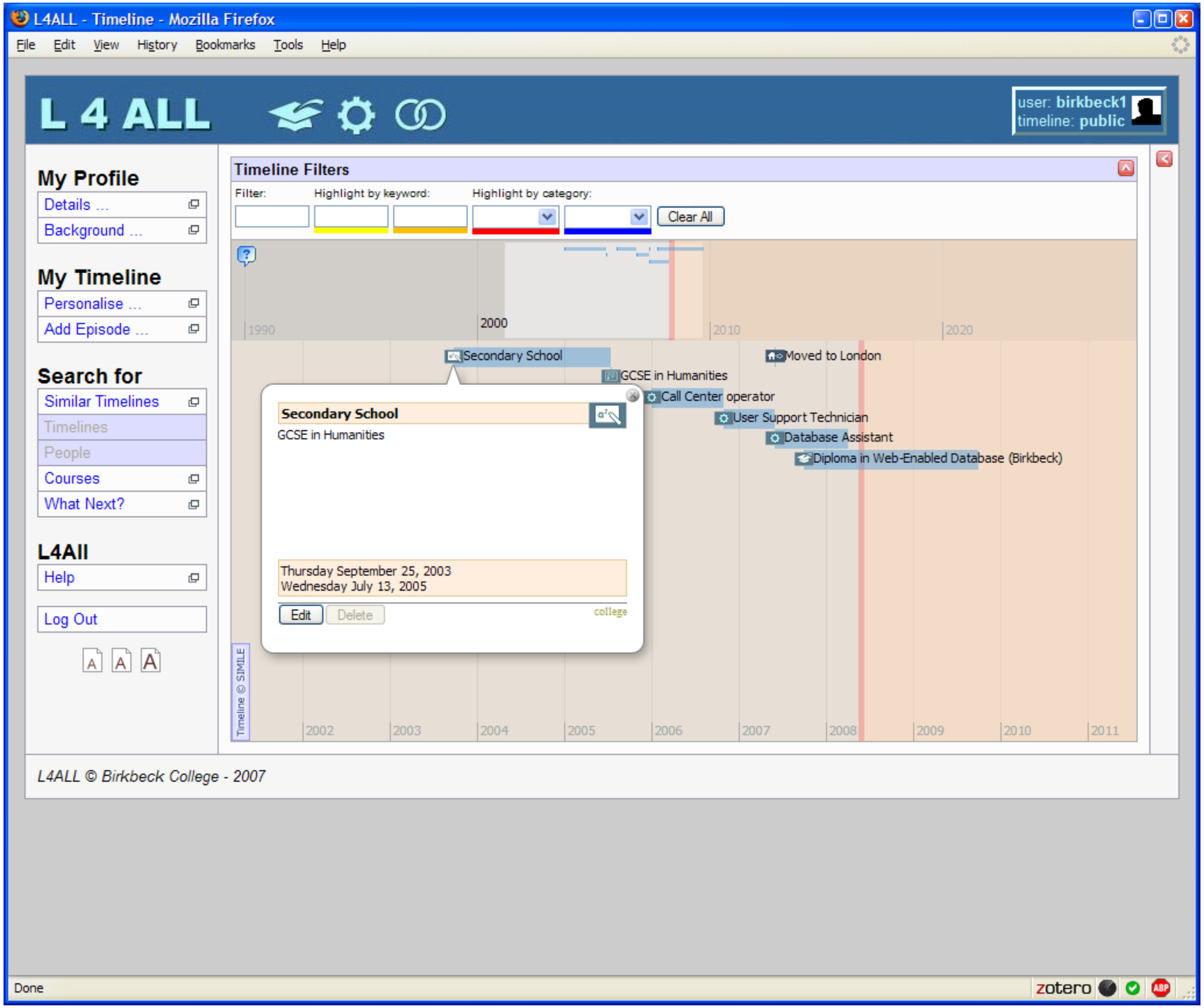


?E1	?E1,RD	?E1,?E2,ED	?E2,?P	?E2,?Goal,ED	?Goal	?Goal,RD	?E2,P,D
ep31	*ep31,4*	*ep31,ep32,1*	*ep32,W*	ep31,ep32,1	ep32	*ep33,2*	*ep32,W,8*
		ep32,ep33,1	ep33,OE	*ep32,ep33,1*	*ep33*	ep32,4	
		ep31,ep33,2		ep31,ep33,2			

5. The ApproxRelax Prototype

- The search facility of the original L4All system for lifelong learners is rather limited:
  - it offers a fixed set of similarity metrics over the timeline data
  - applied to just one level of detail of the classifications of selected categories of episode in the search query
- We have built a prototype called ApproxRelax that employs query approximation and relaxation techniques to provide a more flexible search facility for lifelong learners

Main timeline construction & search screen (L4All)



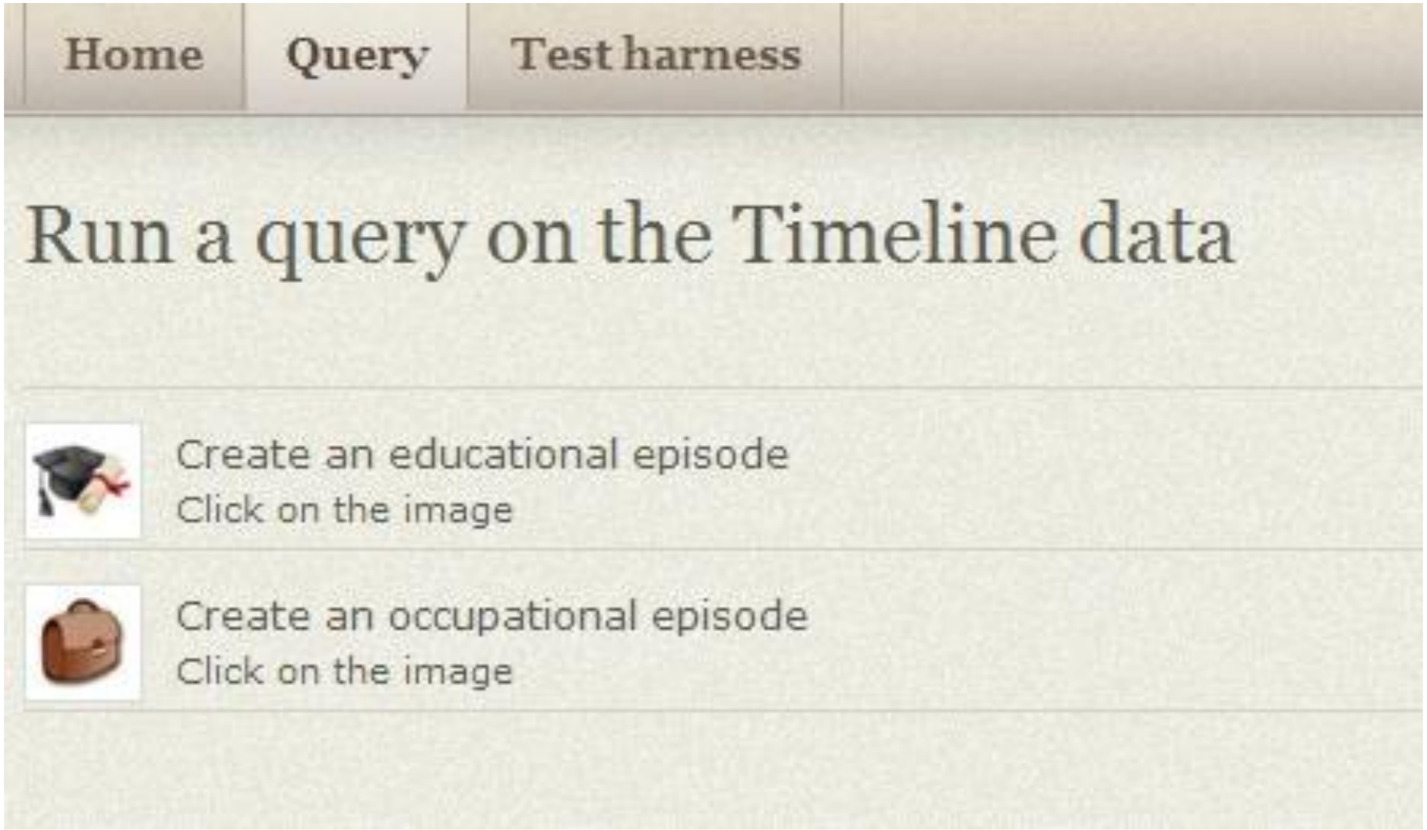
Secondary School

GCSE in Humanities

Thursday September 25, 2003
Wednesday July 13, 2005

L4ALL © Birkbeck College - 2007

Creating a query in ApproxRelax

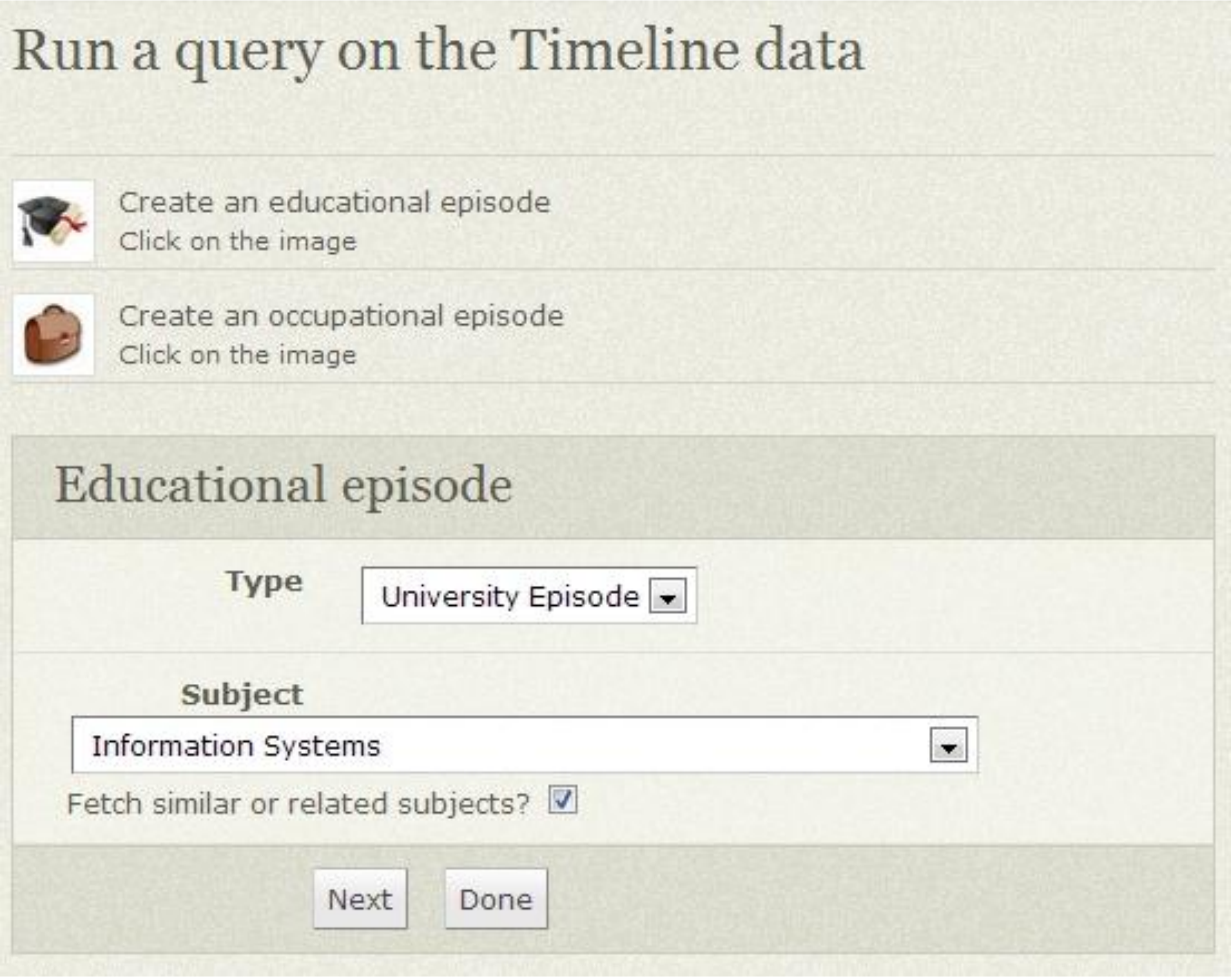


Run a query on the Timeline data

Create an educational episode
Click on the image

Create an occupational episode
Click on the image

Creating a query in ApproxRelax



Run a query on the Timeline data

Create an educational episode
Click on the image

Create an occupational episode
Click on the image

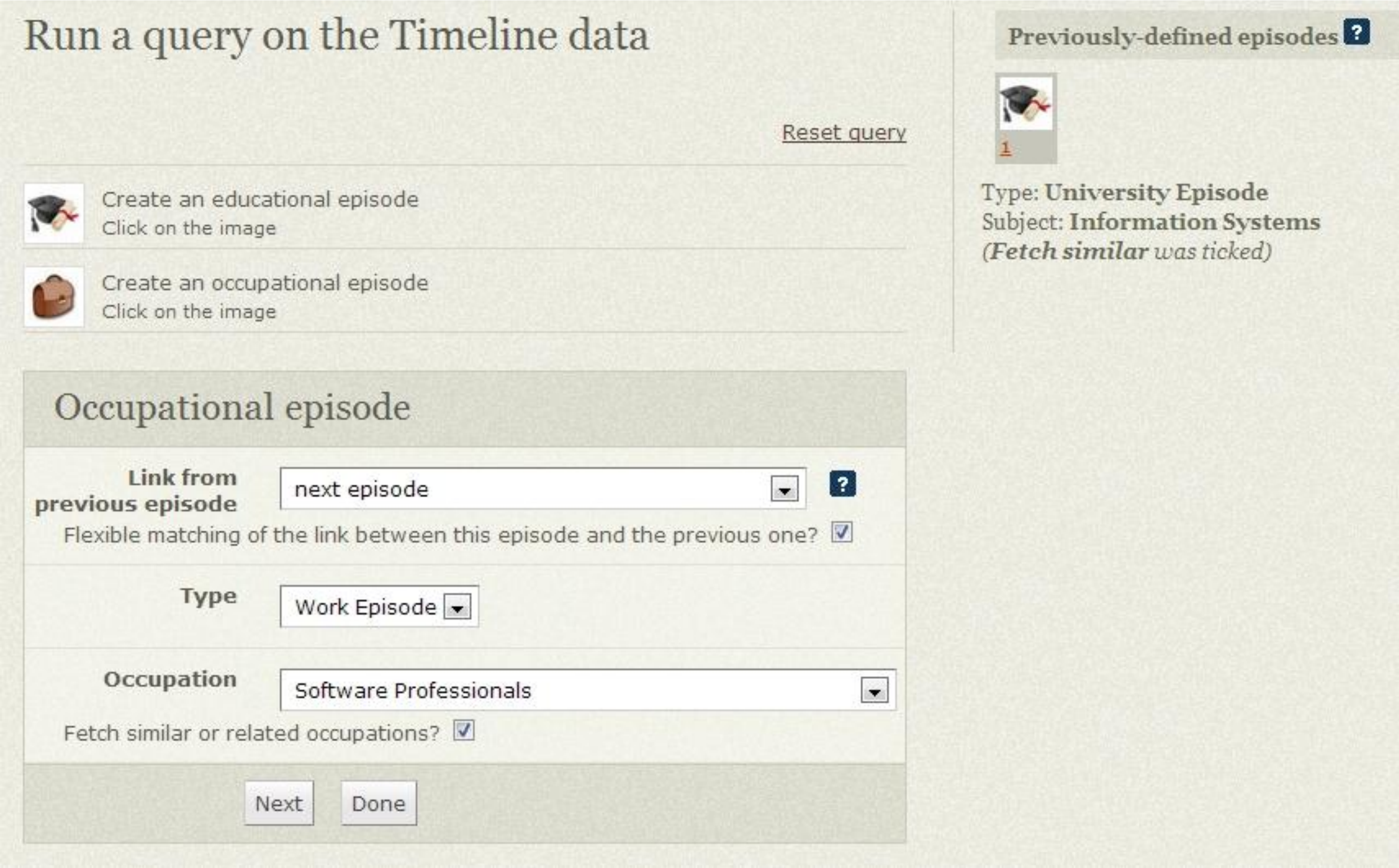
Educational episode

Type University Episode

Subject
Information Systems

Fetch similar or related subjects? ☑

Creating a query in ApproxRelax



Run a query on the Timeline data

Reset query

Create an educational episode
Click on the image

Create an occupational episode
Click on the image

Occupational episode

Link from previous episode: next episode ▾ ?

Flexible matching of the link between this episode and the previous one? ☑

Type: Work Episode ▾

Occupation: Software Professionals ▾

Fetch similar or related occupations? ☑

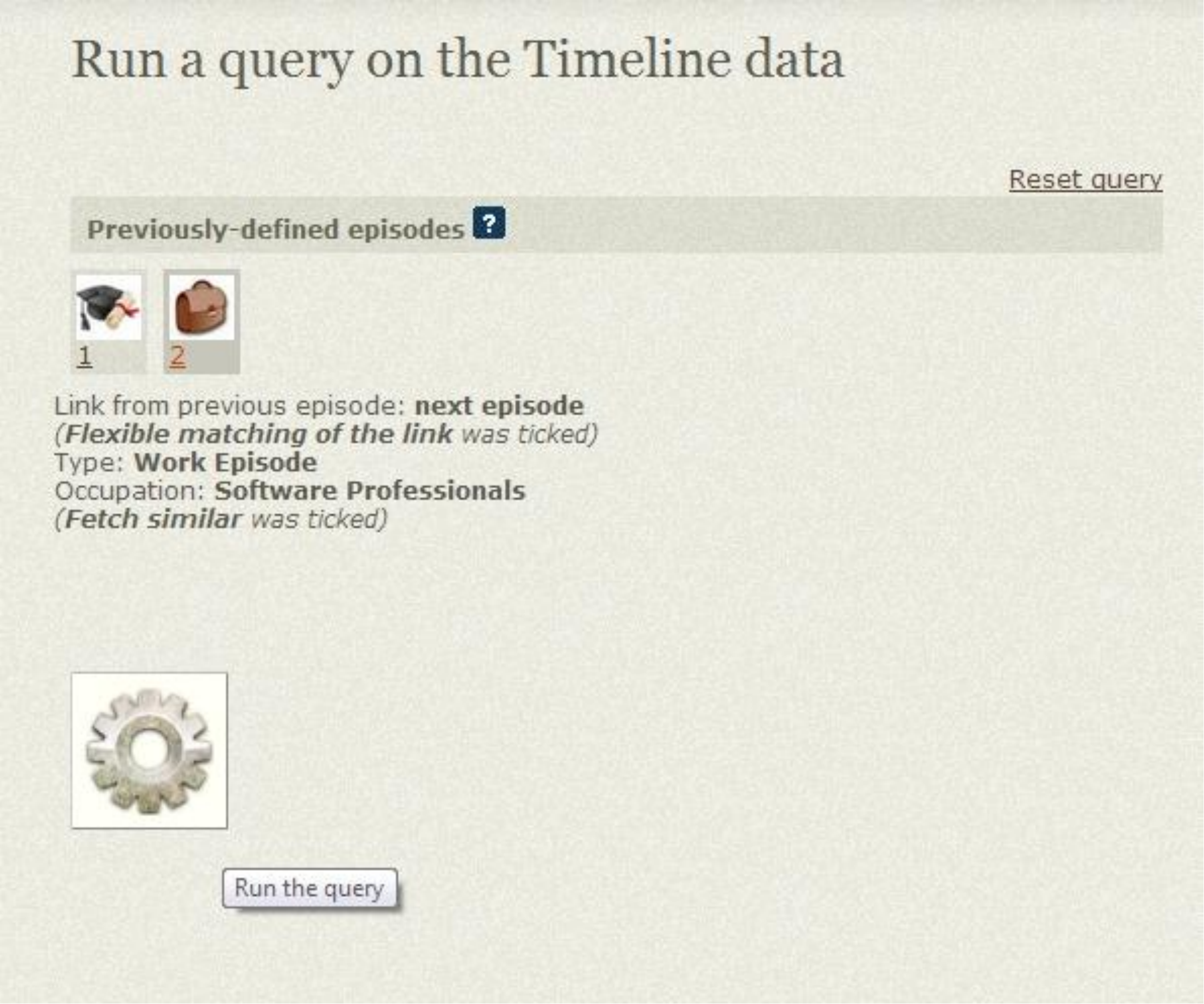
Next Done

Previously-defined episodes ?

1

Type: **University Episode**
Subject: **Information Systems**
*(**Fetch similar** was ticked)*

Viewing your query

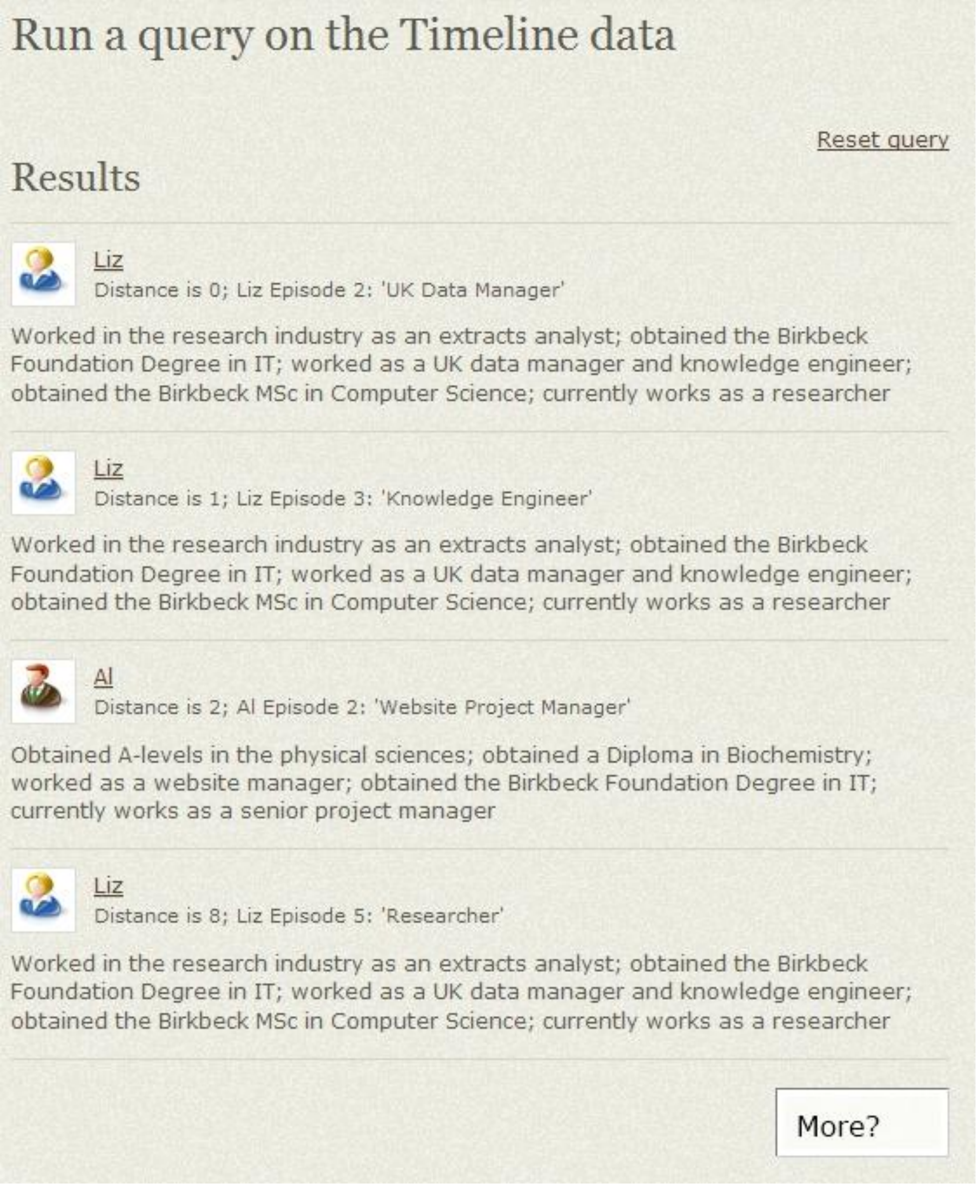


Run a query on the Timeline data

Previously-defined episodes ?

Link from previous episode: **next episode**
*(**Flexible matching of the link** was ticked)*
Type: **Work Episode**
Occupation: **Software Professionals**
*(**Fetch similar** was ticked)*

Viewing the query results



Run a query on the Timeline data

Reset query

Results

Liz
Distance is 0; Liz Episode 2: 'UK Data Manager'

Worked in the research industry as an extracts analyst; obtained the Birkbeck Foundation Degree in IT; worked as a UK data manager and knowledge engineer; obtained the Birkbeck MSc in Computer Science; currently works as a researcher

Liz
Distance is 1; Liz Episode 3: 'Knowledge Engineer'

Worked in the research industry as an extracts analyst; obtained the Birkbeck Foundation Degree in IT; worked as a UK data manager and knowledge engineer; obtained the Birkbeck MSc in Computer Science; currently works as a researcher

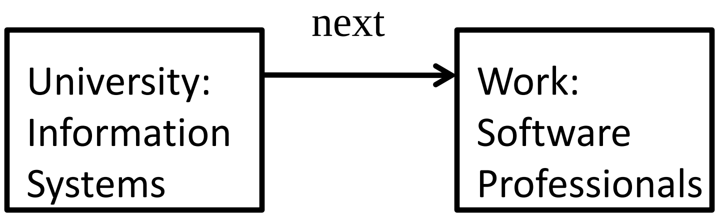
Al
Distance is 2; Al Episode 2: 'Website Project Manager'

Obtained A-levels in the physical sciences; obtained a Diploma in Biochemistry; worked as a website manager; obtained the Birkbeck Foundation Degree in IT; currently works as a senior project manager

Liz
Distance is 8; Liz Episode 5: 'Researcher'

Worked in the research industry as an extracts analyst; obtained the Birkbeck Foundation Degree in IT; worked as a UK data manager and knowledge engineer; obtained the Birkbeck MSc in Computer Science; currently works as a researcher

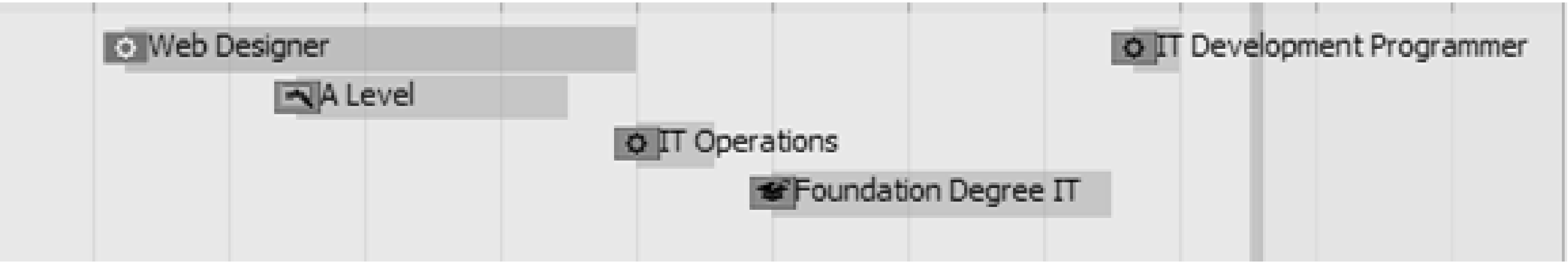
Viewing your query and a selected timeline together



University: Information Systems

next

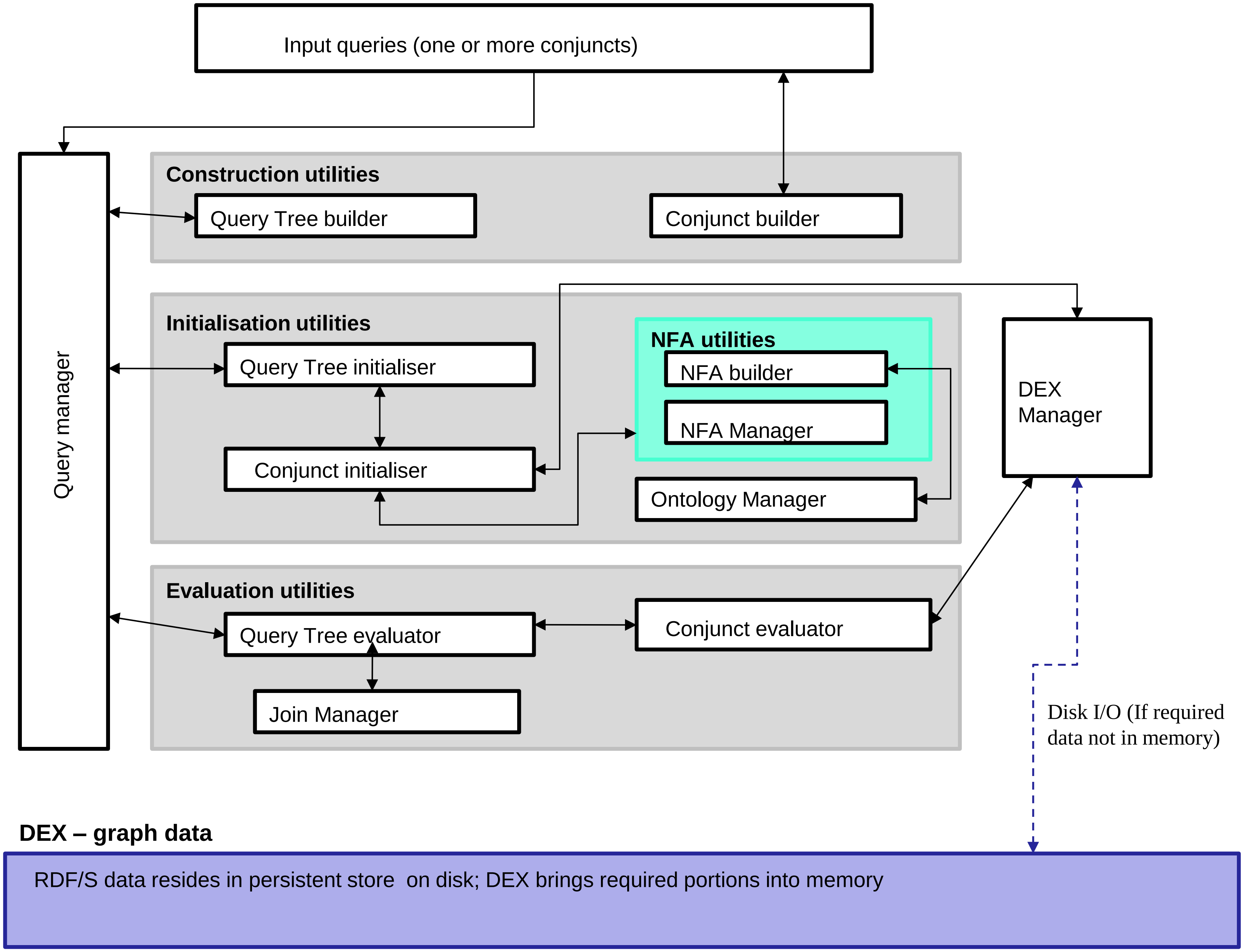
Work: Software Professionals



ApproxRelax Implementation and Evaluation

- Our IEEE Trans. on Learning Technologies paper gives details of the ApproxRelax prototype, including discussion of motivation, implementation and user evaluation of the new query facilities

- Since that work, the ApproxRelax system has been ported to the DEX graph database (www.sparsity-technologies.com/dex)
  - native database connectivity from the ApproxRelax application (which is implemented in C#)
  - indexing capabilities over a native graph-structured data storage format
  - better performance for edge traversal on relevant benchmarks


Input queries (one or more conjuncts)
Query manager
Construction utilities
Query Tree builder
Conjunct builder
Initialisation utilities
Query Tree initialiser
Conjunct initialiser
NFA utilities
NFA builder
NFA Manager
Ontology Manager
DEX
Manager
Evaluation utilities
Query Tree evaluator
Conjunct evaluator
Join Manager
Disk I/O (If required
data not in memory)
DEX – graph data
RDF/S data resides in persistent store on disk; DEX brings required portions into memory

6. Combining APPROX and RELAX into FLEX

- We are currently merging the APPROX and RELAX operators into one integrated FLEX operator, e.g., for the earlier Query 4:

```
(?E2,?P) ← (?E1,type,University),
      FLEX (?E1,qualif.type,EnglishStudies),
       FLEX (?E1,prereq+,?E2), (?E2,job.type,?P),
       FLEX (?E2,prereq+,?Goal), (?Goal,type,Work)
       FLEX (?Goal,job.type,AssistantEditor)
```

- FLEX applies concurrently both approximation and relaxation to a query conjunct, building on our common NFA-based approach both of these
- We term an operation which is either an edit operation or a direct relaxation with respect to an ontology K a *flex* operation
- We obtain the same results as for the earlier Query 4 (and possibly more, depending on structure of the ontology)

Advantages of FLEX

- Allows greater ease of querying for users/implementors, in that they do not need to be aware of the ontology structure and to identify explicitly which parts of their overall query are amenable to relaxation and which to approximation
- Also, if ontology changes, user/implementor does not have to revisit existing queries and check if APPROX and RELAX are still being used appropriately

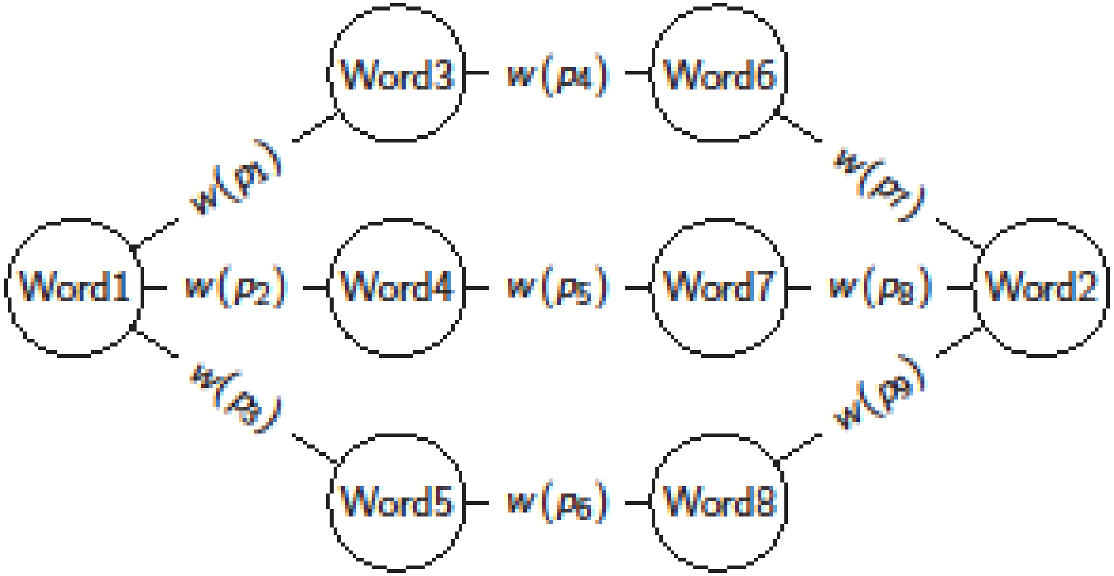
7. Extending with Text Similarity Measures

- Linked Open Data includes many thesauri, dictionaries, encyclopaedias e.g. WordNet and Wiktionary for word meanings, DBPedia for people, places, organisations etc.
- The RDFS vocabulary includes properties that can be used to link resources with human-readable text e.g. rdfs:comment, rdfs:label
- Specific resources may refine these into subclasses, such as hasAbstract in DBPedia, hasGloss[ary] in YAGO
- We would like to make use of this text information in supporting users to finding relevant resources
- We make use of “meta-properties” in dictionaries or thesauri, such as synonym, antonym, hypernym, in order to make semantic connections between search text in a query and text information in the data

Extending with Text Similarity Measures

- We take into account the path distance between words and the weights associated with the properties connecting them
  - each property has a weight
  - two words may be connected through multiple paths
  - we calculate a weighted average over all paths, filtering out paths beyond a certain length
- This approach is different from a corpus-based approach, in that we are exploiting the potential of RDF:
  - corpus-based approaches compute word similarity based on the co-occurrence in a set of documents
  - our approach does not use a set of documents, only dictionaries/thesauri encoded in RDF
  - we use simple SPARQL queries to find paths between words

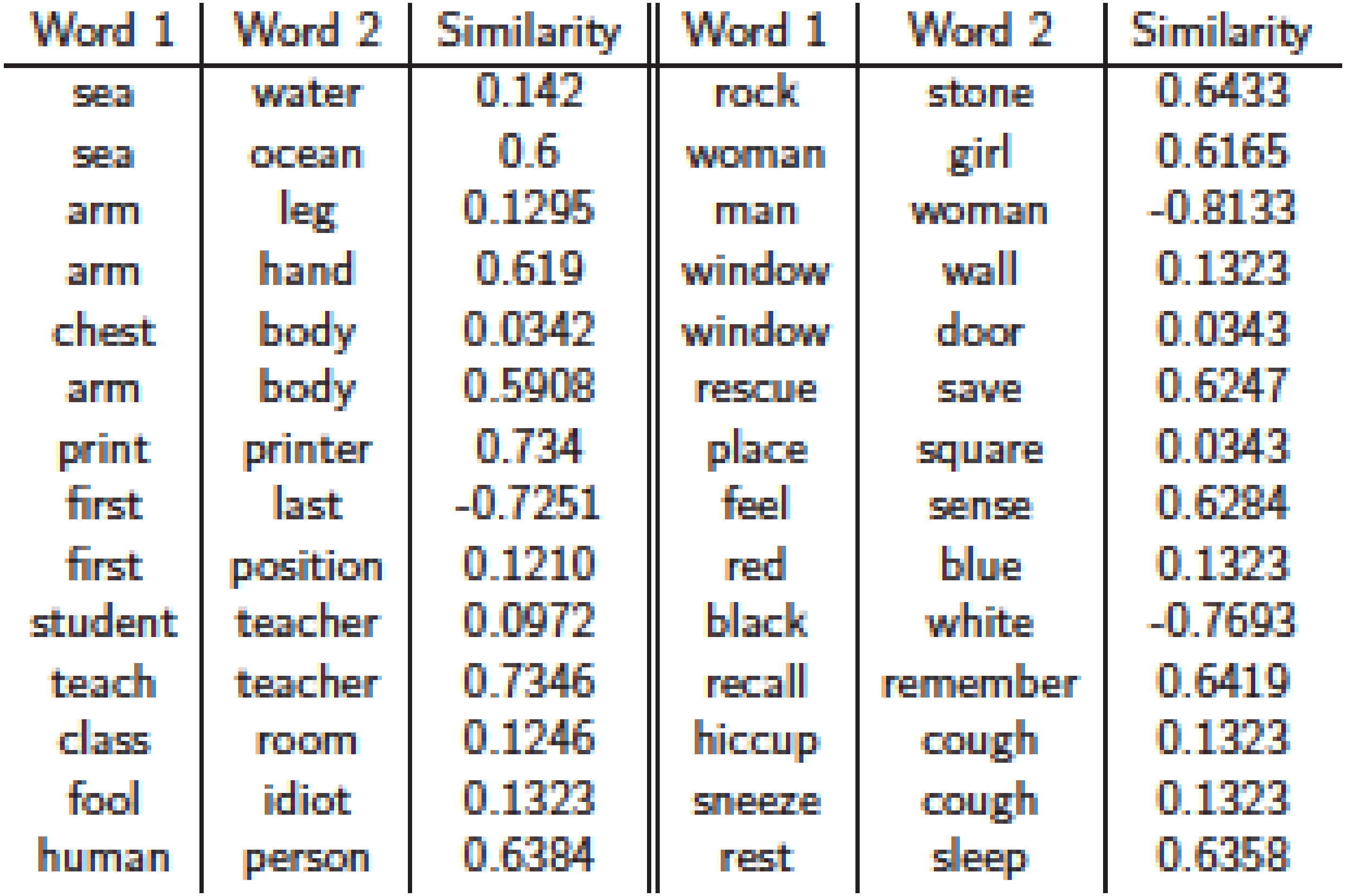
Extending with Text Similarity Measures



Word similarity function, where α_i and w(p) are weights:

$$F_g^n(x,y) = \alpha_1 \frac{\sum_{p \in pred(x,y)} w(p)}{|pred(x,y)|} + \sum_{j=2}^{n} \alpha_j \frac{\sum_{t \in paths^j(x,y)} \prod_{(s,p,o) \in t} w(p)}{|paths^j(x,y)|}$$

Some preliminary results



Word 1	Word 2	Similarity	Word 1	Word 2	Similarity
sea	water	0.142	rock	stone	0.6433
sea	ocean	0.6	woman	girl	0.6165
arm	leg	0.1295	man	woman	-0.8133
arm	hand	0.619	window	wall	0.1323
chest	body	0.0342	window	door	0.0343
arm	body	0.5908	rescue	save	0.6247
print	printer	0.734	place	square	0.0343
first	last	-0.7251	feel	sense	0.6284
first	position	0.1210	red	blue	0.1323
student	teacher	0.0972	black	white	-0.7693
teach	teacher	0.7346	recall	remember	0.6419
class	room	0.1246	hiccup	cough	0.1323
fool	idiot	0.1323	sneeze	cough	0.1323
human	person	0.6384	rest	sleep	0.6358

Extending with Text Similarity Measures

- The above metric can be extended to sentences and text snippets in several ways e.g. bag-of-words, sequence-of-words, comparison of the grammatical structure of sentences

- Initially we plan to incorporate these facilities into a tractable subset of SPARQL 1.1, including *property path queries* (similar to regular path queries)

- Next, we will also incorporate query approximation and relaxation, similarly to the ApproxRelax prototype

An example query, against YAGO

Suppose I am a History of Science researcher and wish to find scientists born in London. I am also interested in scientists who used to live or were born near London. Also, I don't know exactly how YAGO records that a person is a scientist.

Using FLEX, my results can include scientists who used to live or were born near London. Using similarity matching, results can return resources whose glossary description contains the word “scientist” or similar:

```
PREFIX yago:http://yago-knowledge.org/resource/
SELECT * WHERE {
    FLEX (?p yago:wasBornIn/yago:isLocatedIn yago:London) .
    ?p rdf:type ?c . ?c yago:hasGloss ?string .
    FILTER sim(?string, "scientist”) > 0.7}
```

An example query, against YAGO

Approximation and relaxation : possible substitutions:

wasBornIn → livesIn

isLocatedIn → hasArea

isLocatedIn → hasNeighbor

isLocatedIn → placedIn (subproperty/superproperty relationship)

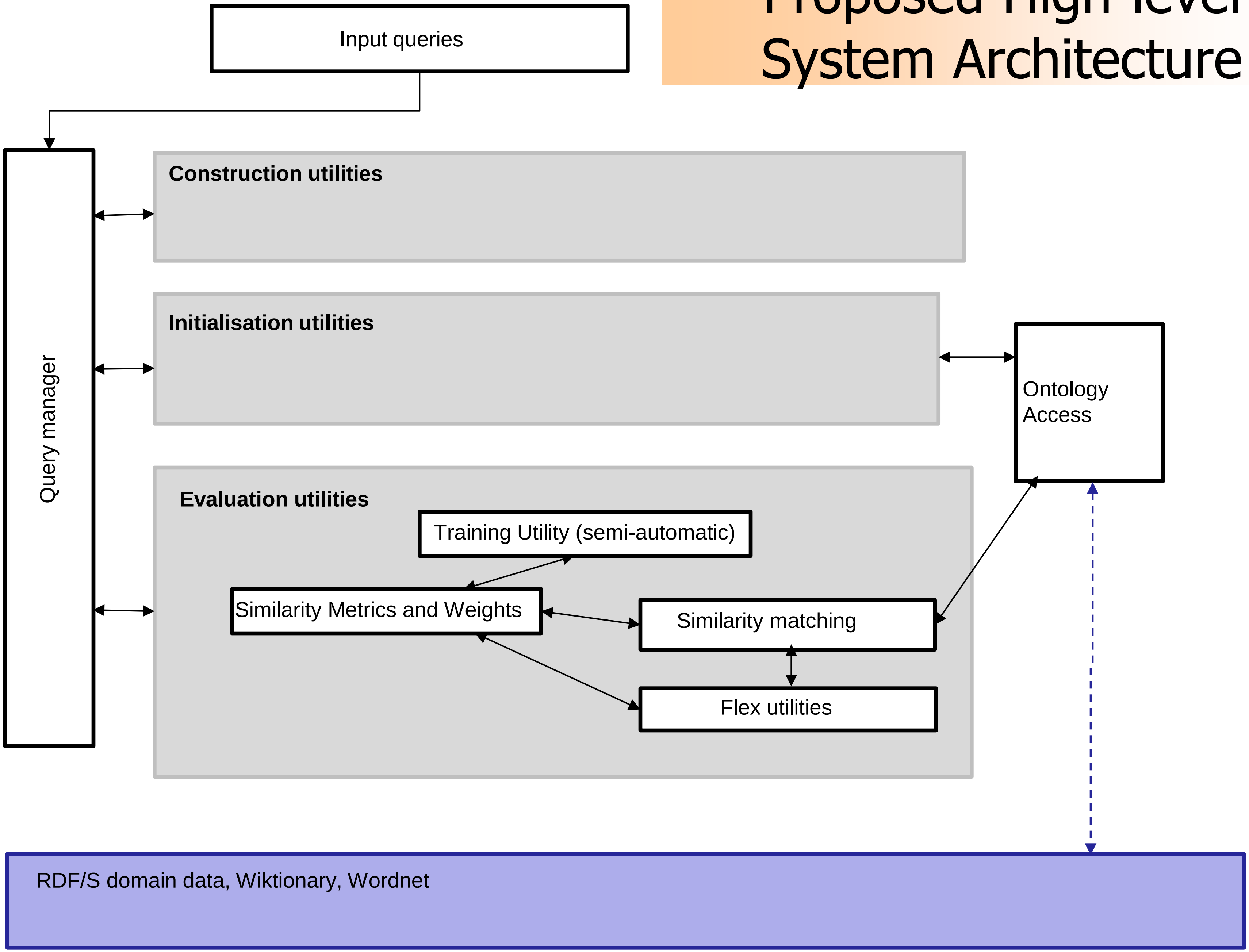
Similarity : possible matches against hasGloss property:

"scientist" → "a person with advanced knowledge of one or more sciences"

"scientist" → "an expert in the science of economics"

"scientist" → "someone trained in computer science and linguistics who uses computers for natural language processing"

Proposed High-level System Architecture



8. Current Work

- Ongoing empirical evaluation of the query processing algorithms in the ApproxRelax prototype
- Design of optimisations e.g.
  - compact representations of the approximated/relaxed automata
  - trade-off between incremental and full evaluation of the product automaton
  - caching partially computed parts
  - materialising parts
  - using techniques such as those of Koschmeider and Leser, SSDBM 2012, to split queries into smaller queries, at labels that are “rare” in the data

Current Work

- Implementing the SPARQL-based system, incorporating FLEX and Text Similarity as well
- Implementing the Training Utility, in which learning methods and user feedback will be used to set the weights and metrics employed, to obtain a more accurate ranking
- Extending the query evaluator to incorporate inference rules and query rewriting, e.g. for capturing
  - context
  - domain knowledge
  - more specialist linguistic information
- Extending to querying distributed RDF data