Combining Approximation and Relaxation in Semantic Web Path Queries

Alex Poulovassilis, Peter Wood
Birkbeck, University of London

ISWC'10, November 2010

Outline of the talk

1. Motivation

- Volumes of semi-structured data available on the web
- Volumes, complexity and heterogeneity of such data means that users may not be aware of its full structure
- Need to be assisted by querying systems that not limited to exact matching of users' queries
- In this paper we investigate
  - combining both *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 general 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

- Our query language is based on *conjunctive regular path queries*:

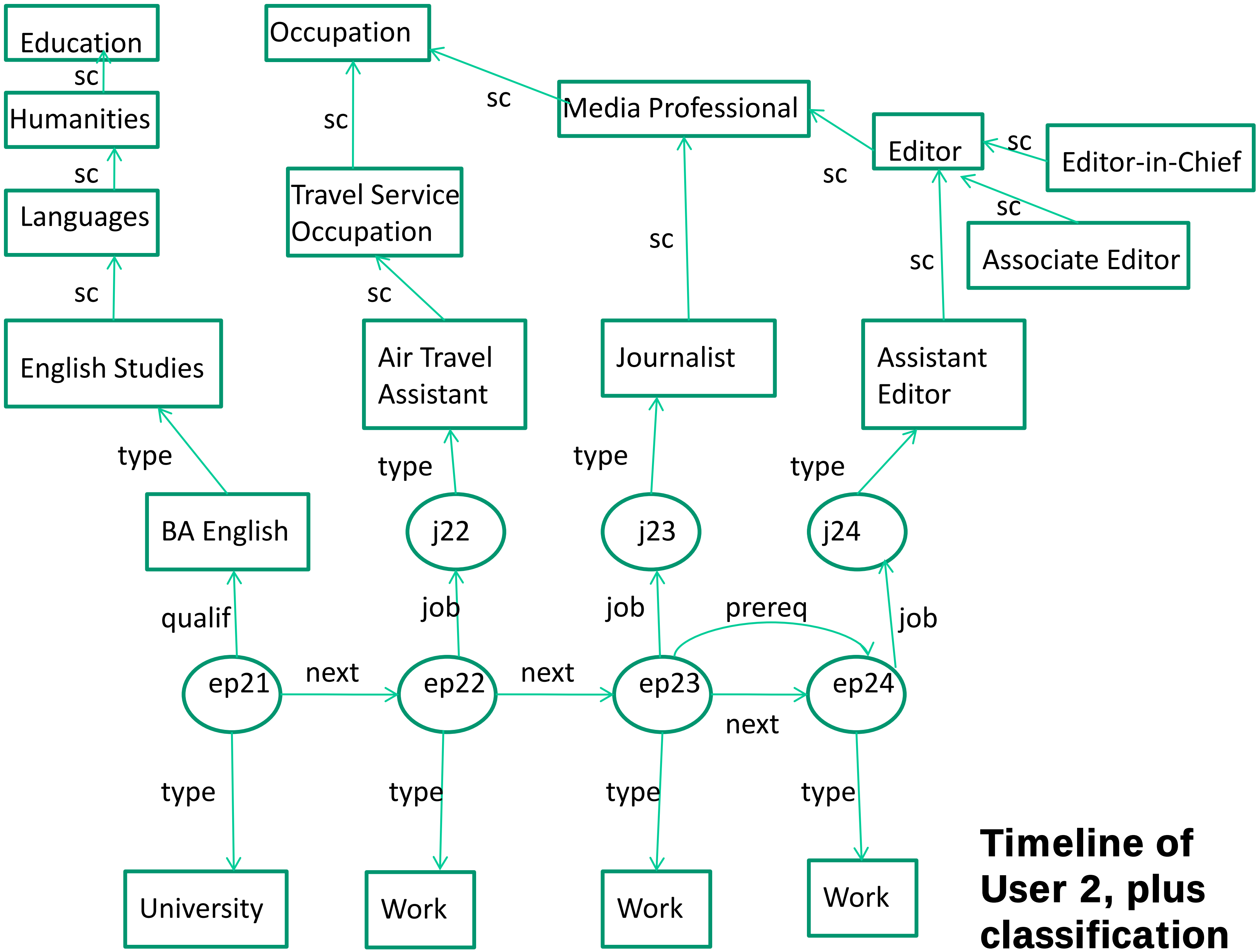
$$(Z_1, \ldots, Z_m) \leftarrow (X_1, R_1, Y_1), \ldots, (X_n, R_n, Y_n)$$

where the X_i , Y_i are variables or constants;

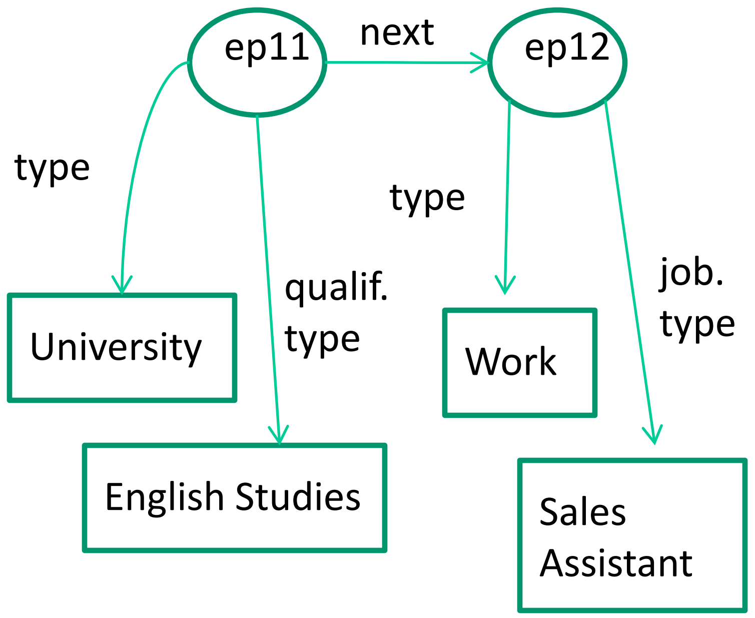
the R_i are *regular expressions* over the alphabet of edge labels, Σ; and the Z_i are variables that also appear in the query body

Example – L4All System

- 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 current 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 are currently investigating how the techniques described in this paper can be used to provide a more flexible search facility for L4All users

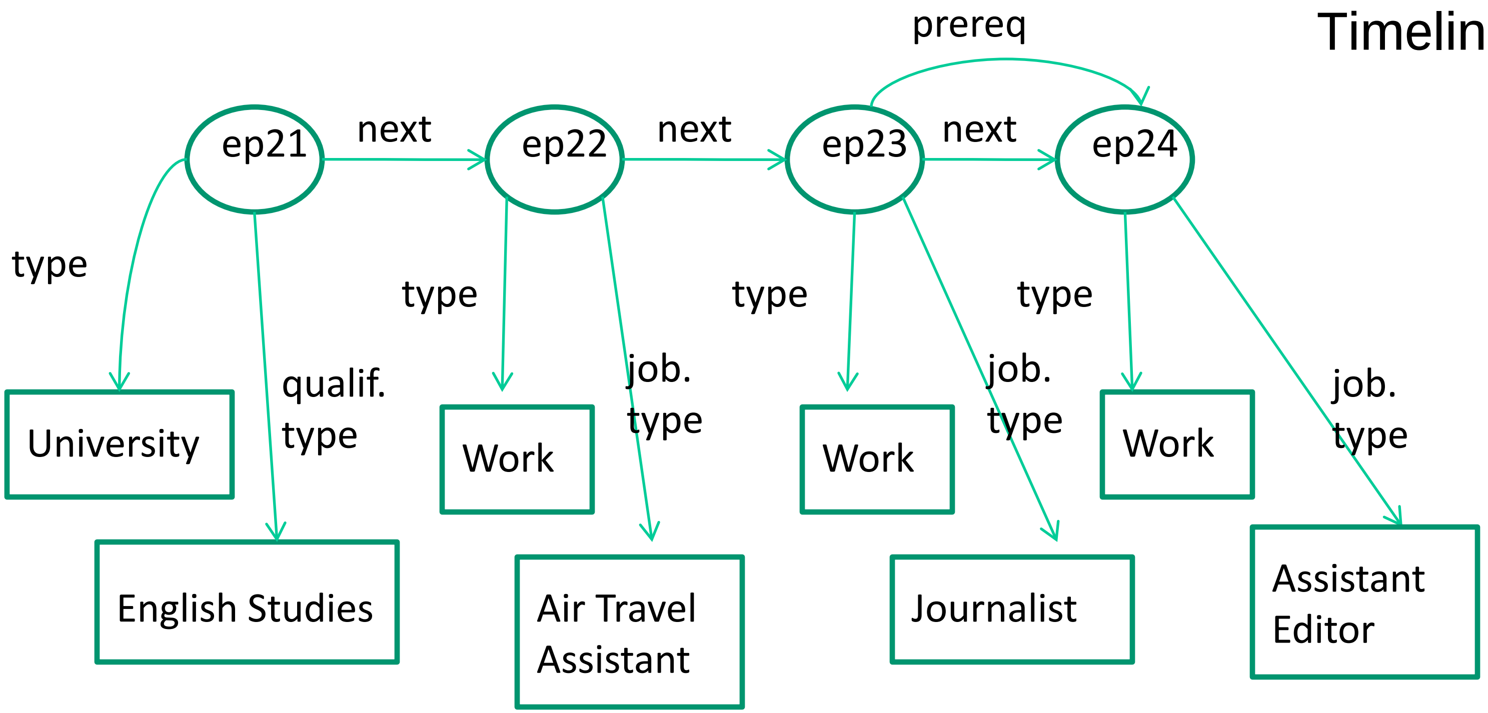


Timeline of User 2, plus classification information



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

Timeline of User 2

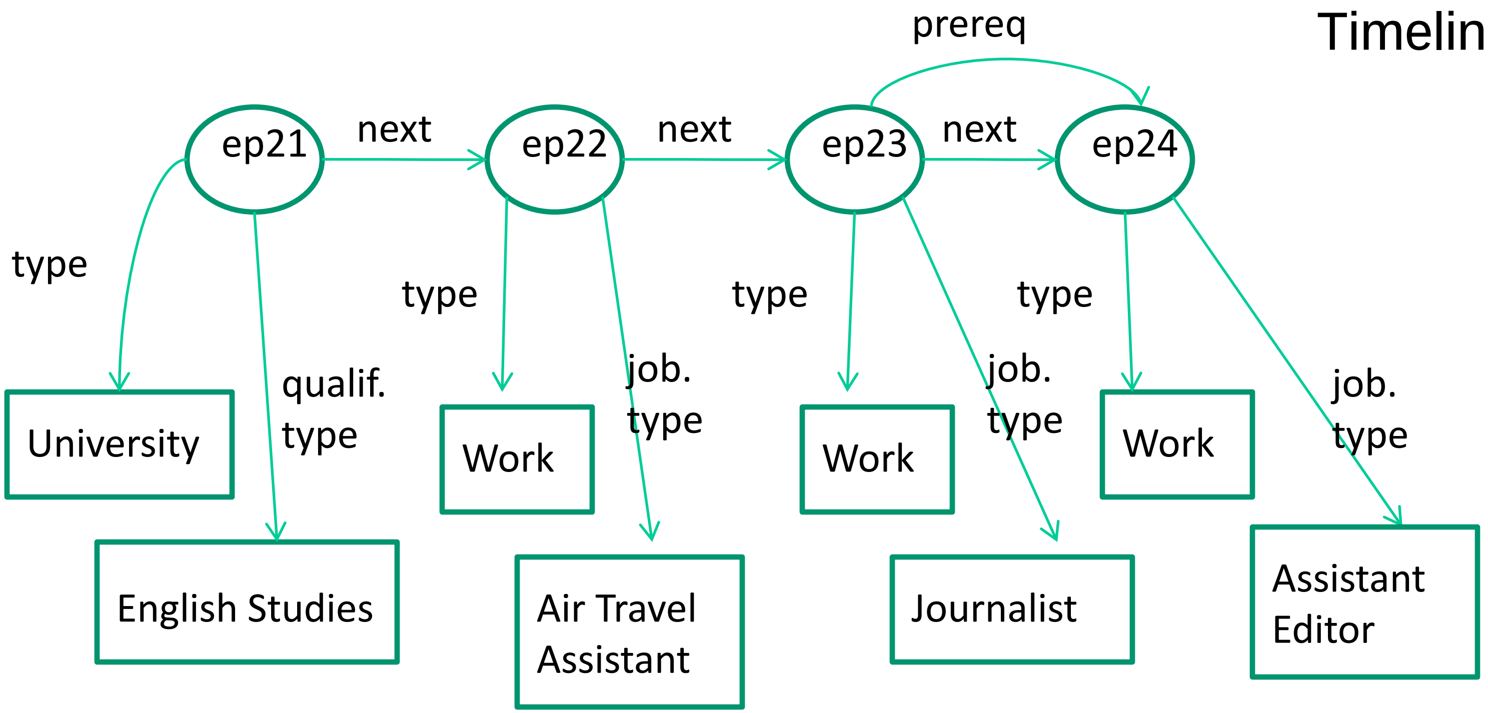


Query 1: I've done this so far. What work positions can I reach and how? E.g. selecting just the relevant prefix of my timeline (my English degree, rather than my temporary work as a Sales Assistant):

(?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

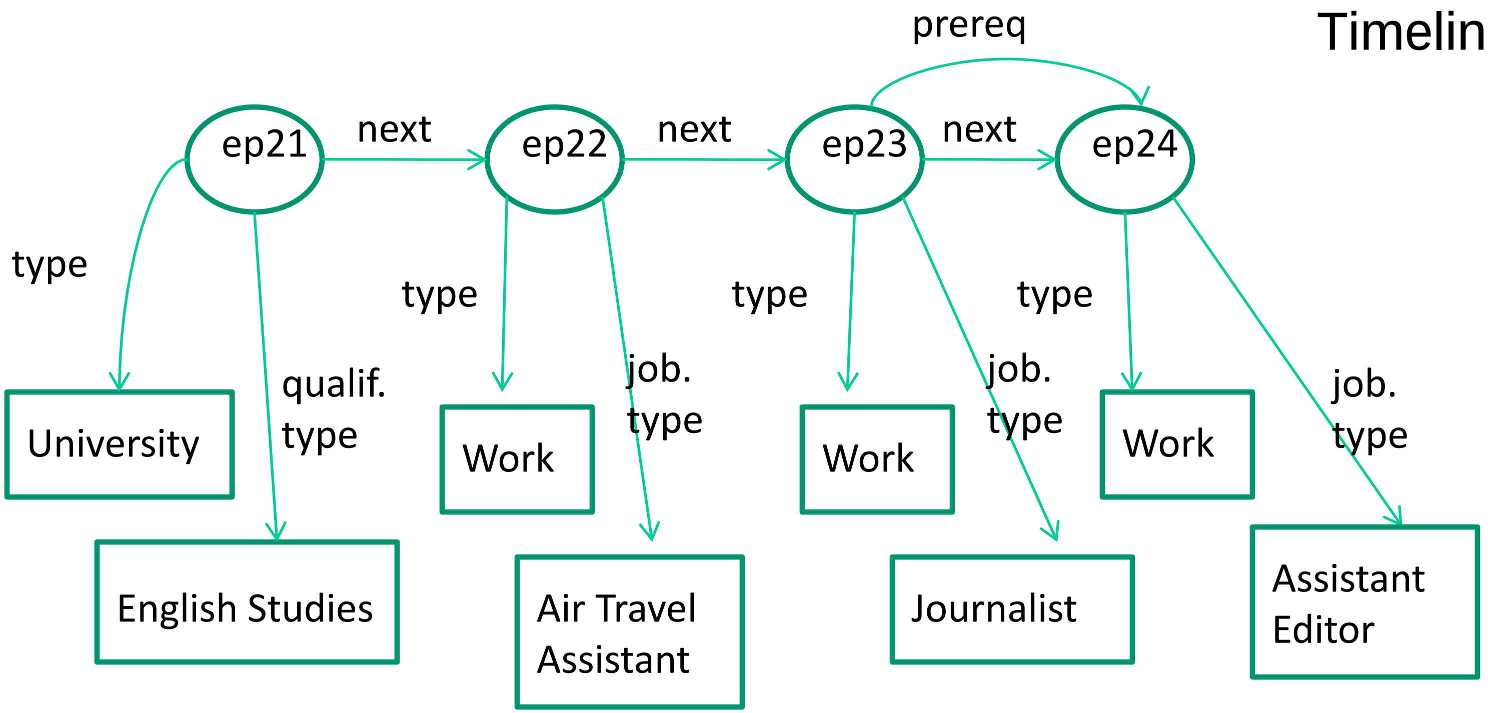
Timeline of User 2



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

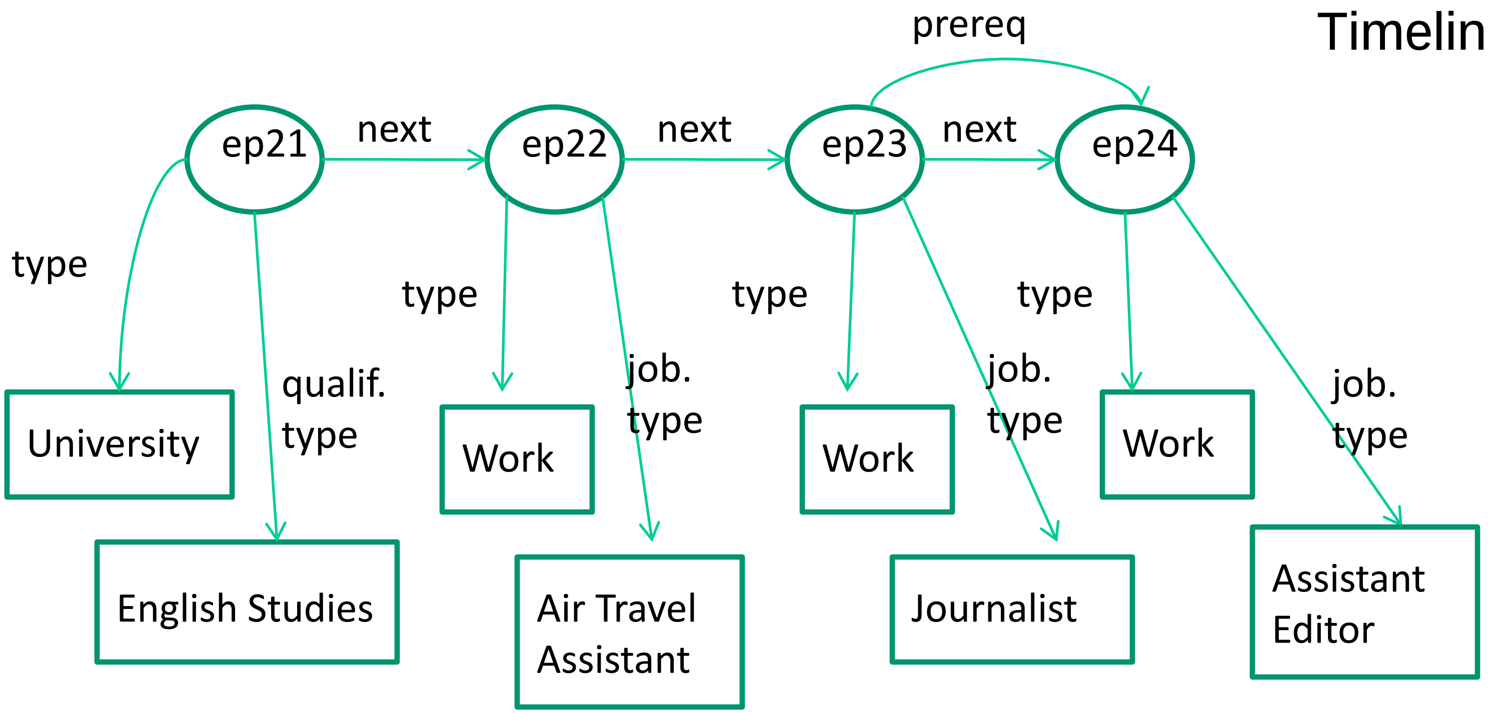


The regular expression prereq+ can be approximated by next.prereq* at edit distance 1. This allows the system to return

(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

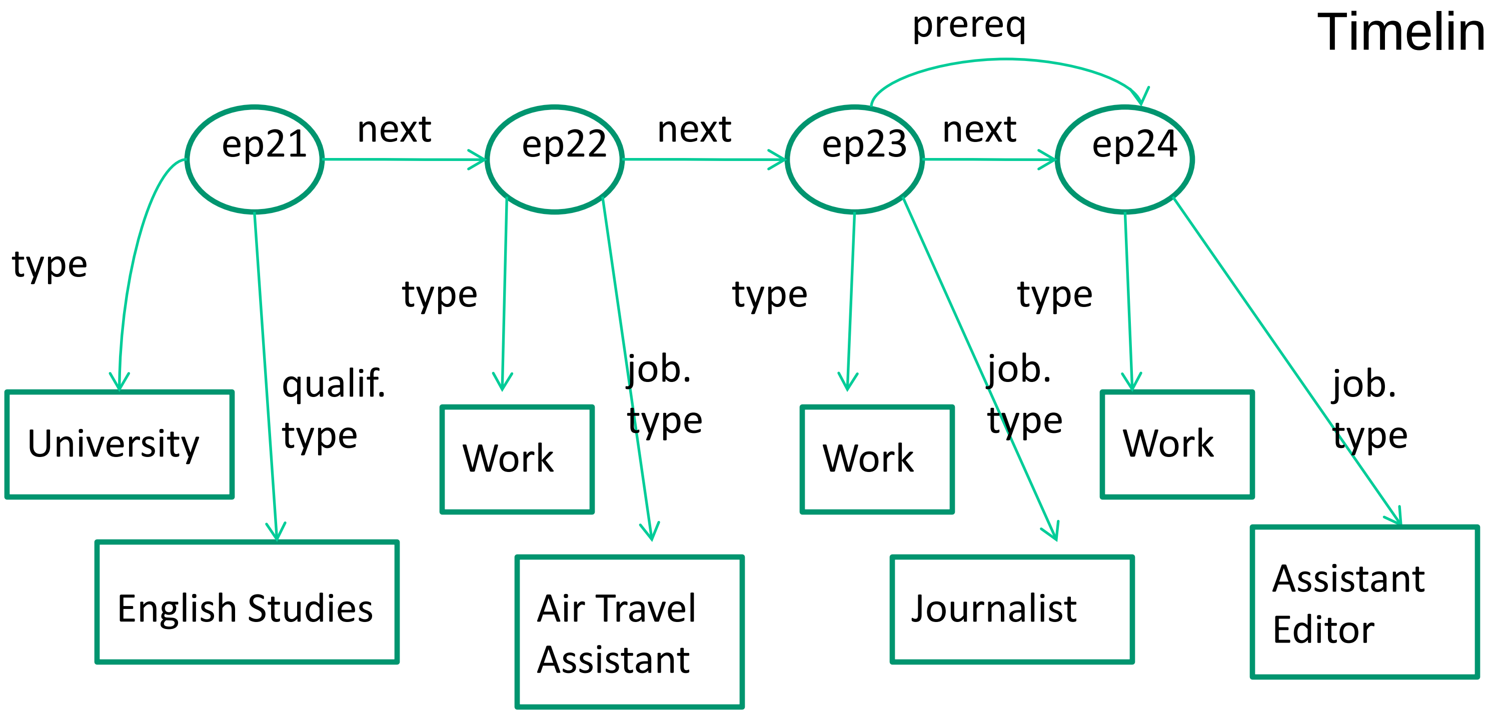


next.prereq* can be approximated by next.next.prereq*, now at edit distance 2. This allows the following answers to be returned:

(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.

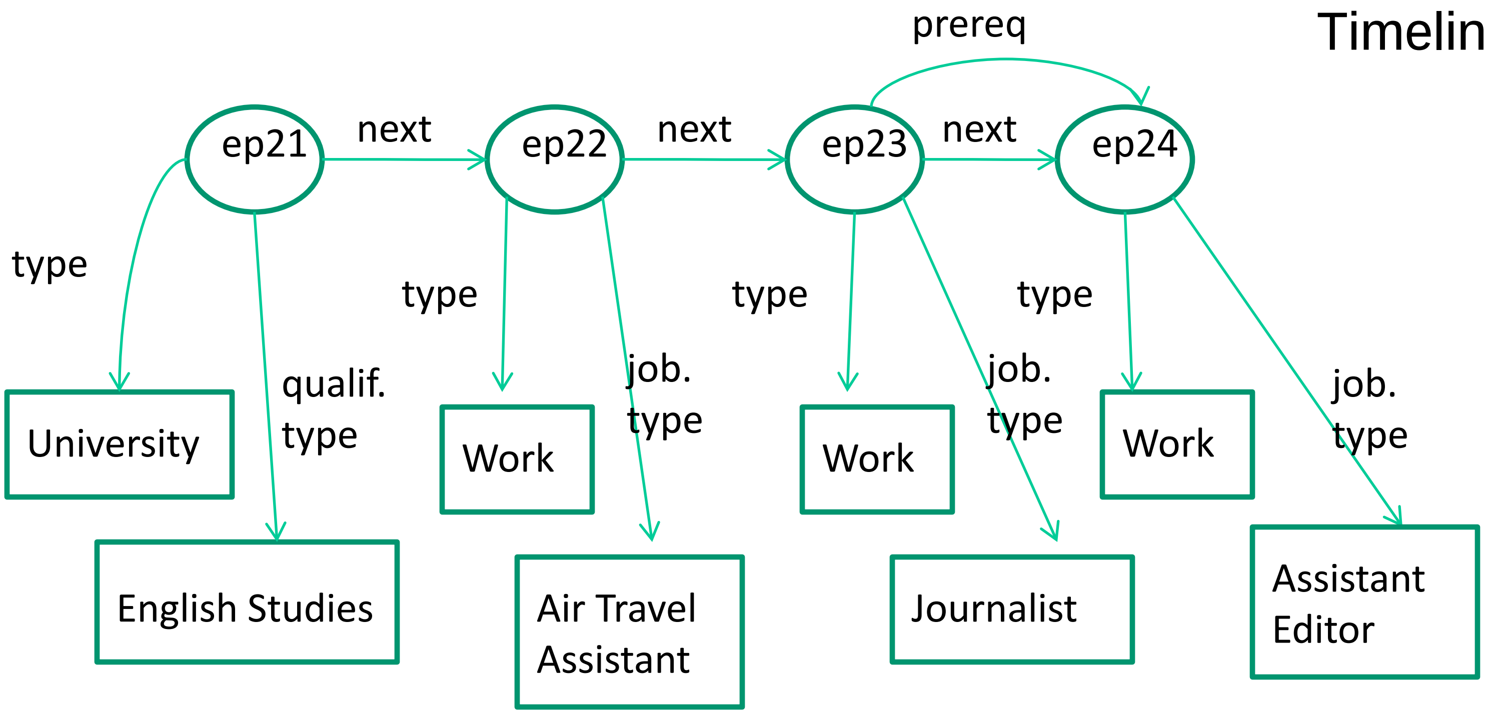
Timeline of User 2



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.

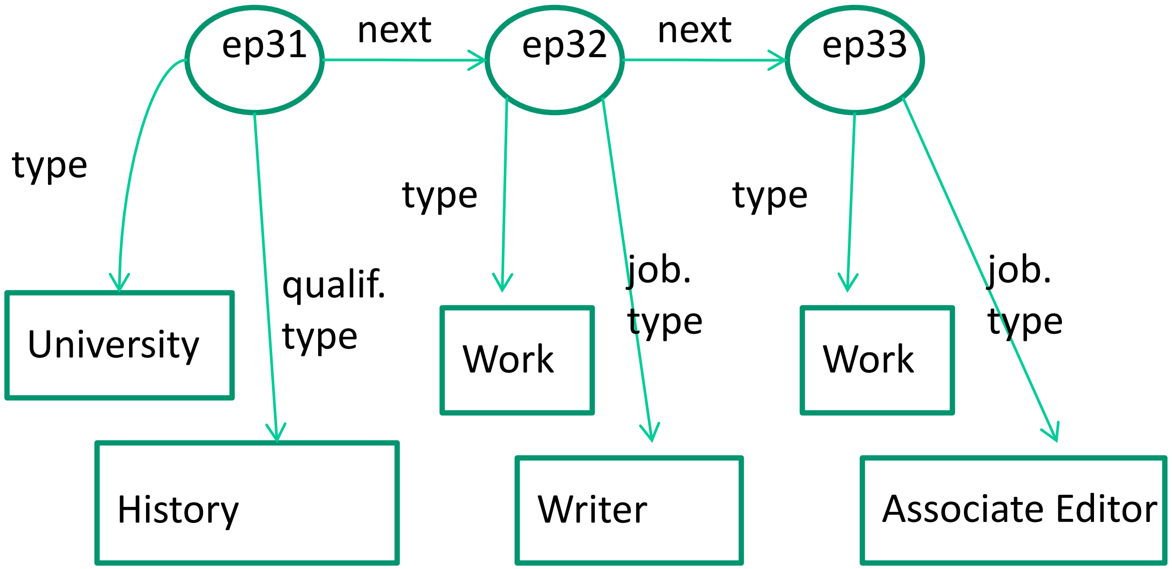
Rather than Query 2:

```
(?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:

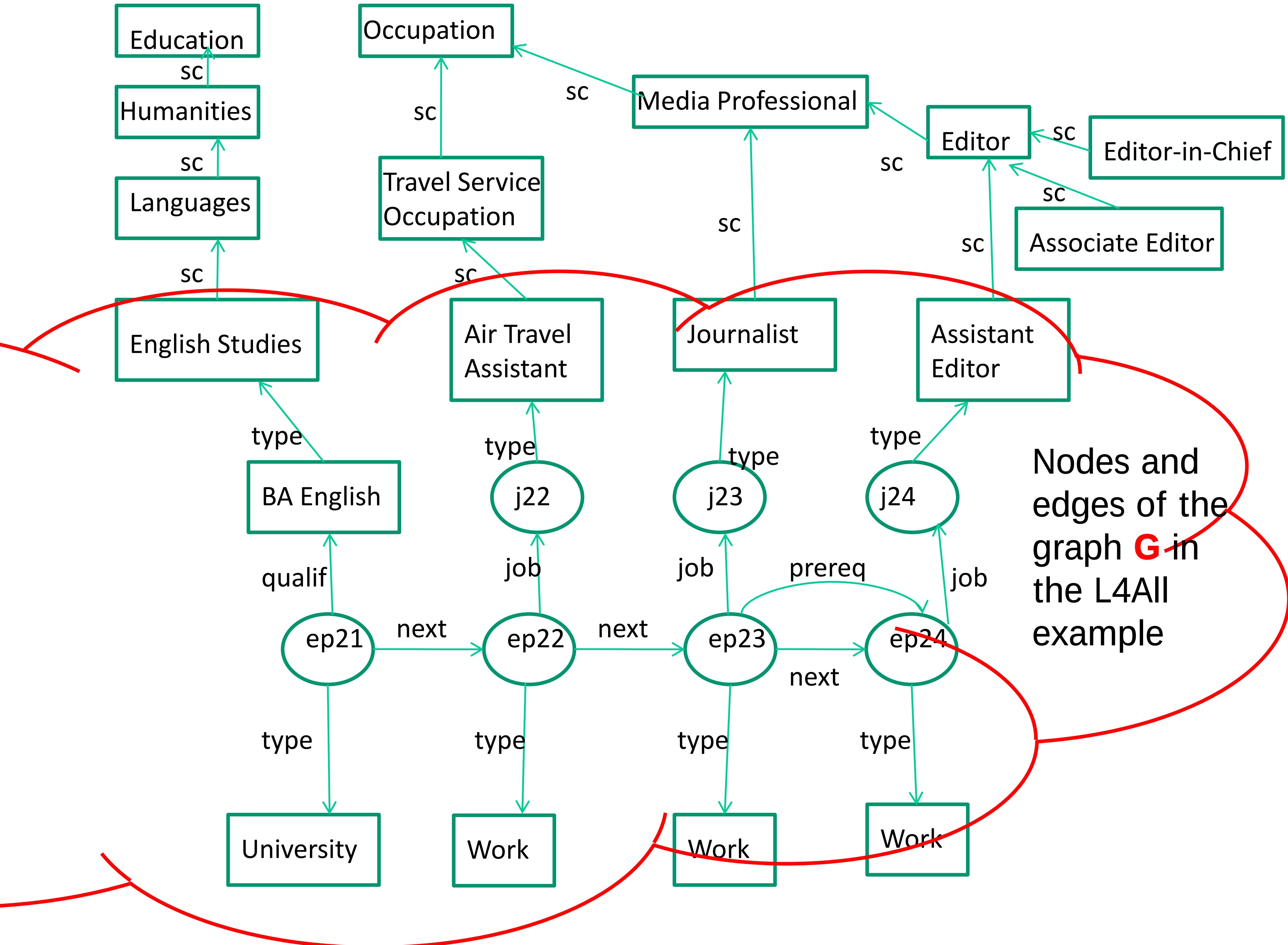
(?E2,?P) ← (?E1,type,University), **RELAX**(?E1,qualif.type,EnglishStudies),
APPROX (?E1,prereq+,?E2), (?E2,job.type,?P),
APPROX (?E2,prereq+,?Goal), (?Goal,type,Work)
RELAX (?Goal,job.type,AssistantEditor)

3. Single-conjunct regular path queries

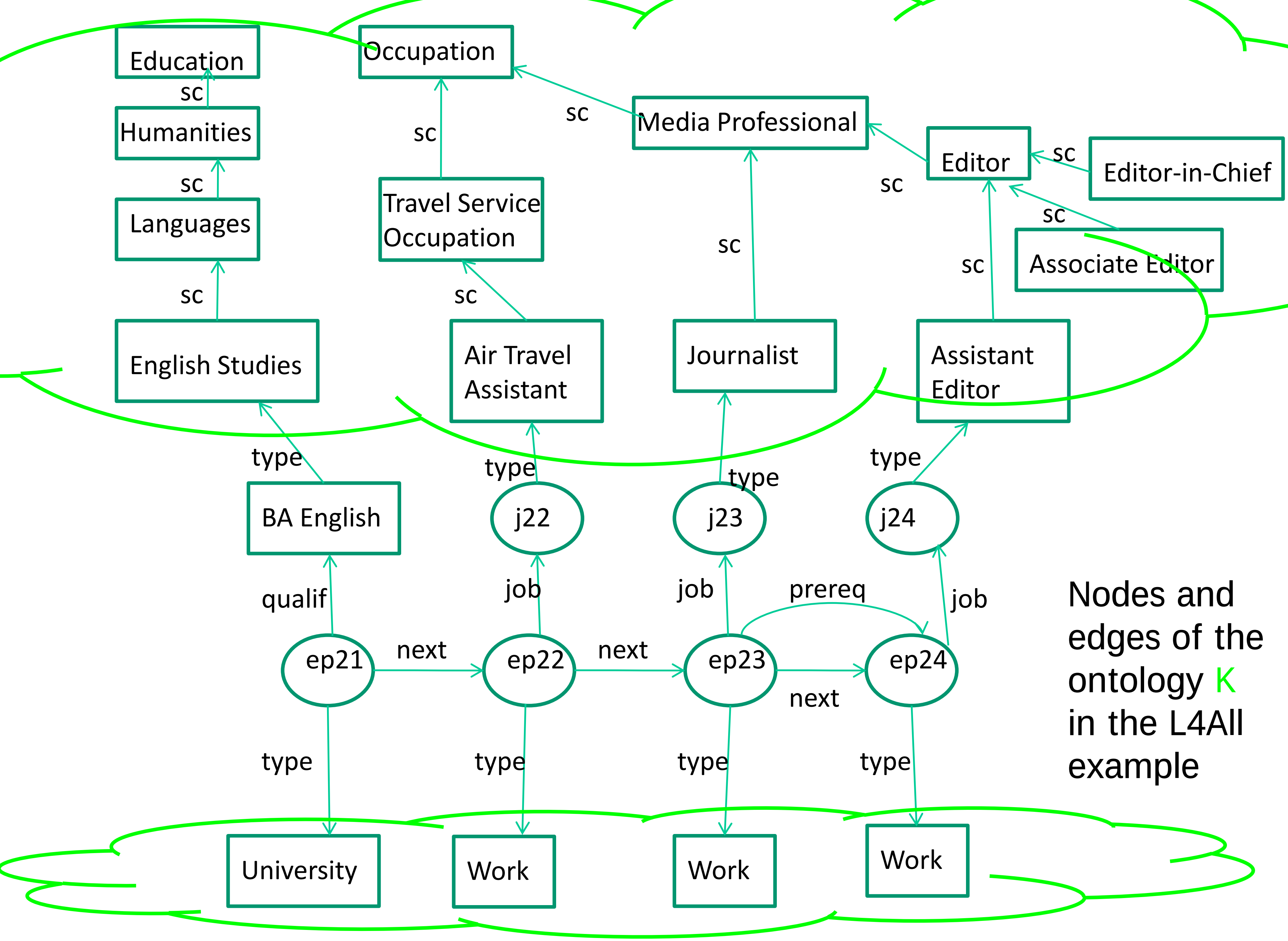
- In the paper we consider a semi-structured data model comprising a directed graph $G = (V,E)$ and an ontology $K = (V_K,E_K)$
- V contains nodes representing entity instances or entity classes, each labelled with a distinct constant
- Each edge in E is labelled with a symbol drawn from a finite alphabet $\Sigma \cup \{type\}$
- 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 L4All example



Nodes and edges of the ontology K in the L4All example

Single-conjunct regular path queries

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

  vars $\leftarrow$ (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 $\Sigma' = \Sigma \cup \{type\}$

- 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, $\mathcal{L}$(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 here are insertions, deletions and substitutions of edge labels, each with an assumed cost of α
  - We envisage the user being able to specify which edit operations should be applied by the system when answering a particular query, or 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 *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.
   - Proposition 1 states the complexity of this traversal as being $O(|R|^2|V| \ (| \Sigma'||E| \ + |V| \log (|R||V|) \ ) \ )$, assuming use of Dijkstra's algorithm

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 (see Figure 2 of the paper) in order to relax regular path queries

- 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 (c,type-,Y) , where the range of a is c (position of Y needs to be preserved, hence use of type-)

- 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 =

  $l_1 l_2 \ \dots \ l_n$

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

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

  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

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

- 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 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 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 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* to return the next query answer, in order of non-decreasing relaxation distance from Q

4. Multi-conjunct queries

- A general query Q is of the form
  $(Z_1, \ldots, Z_m) \leftarrow (X_1, R_1, Y_1), \ldots, (X_j, R_j, Y_j),$
  $\text{APPROX}(X_{j+1}, R_{j+1}, Y_{j+1}), \ldots, \text{APPROX}(X_{j+k}, R_{j+k}, Y_{j+k}),$
  $\text{RELAX}(X_{j+k+1}, R_{j+k+1}, Y_{j+k+1}), \ldots, \text{RELAX}(X_{j+k+n}, R_{j+k+n}, Y_{j+k+n})$

- 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 a weighted sum of
  - the 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 sum of 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*$(\theta, Q) \leq$ *dist*(φ, Q)

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

Multi-conjunct query evaluation

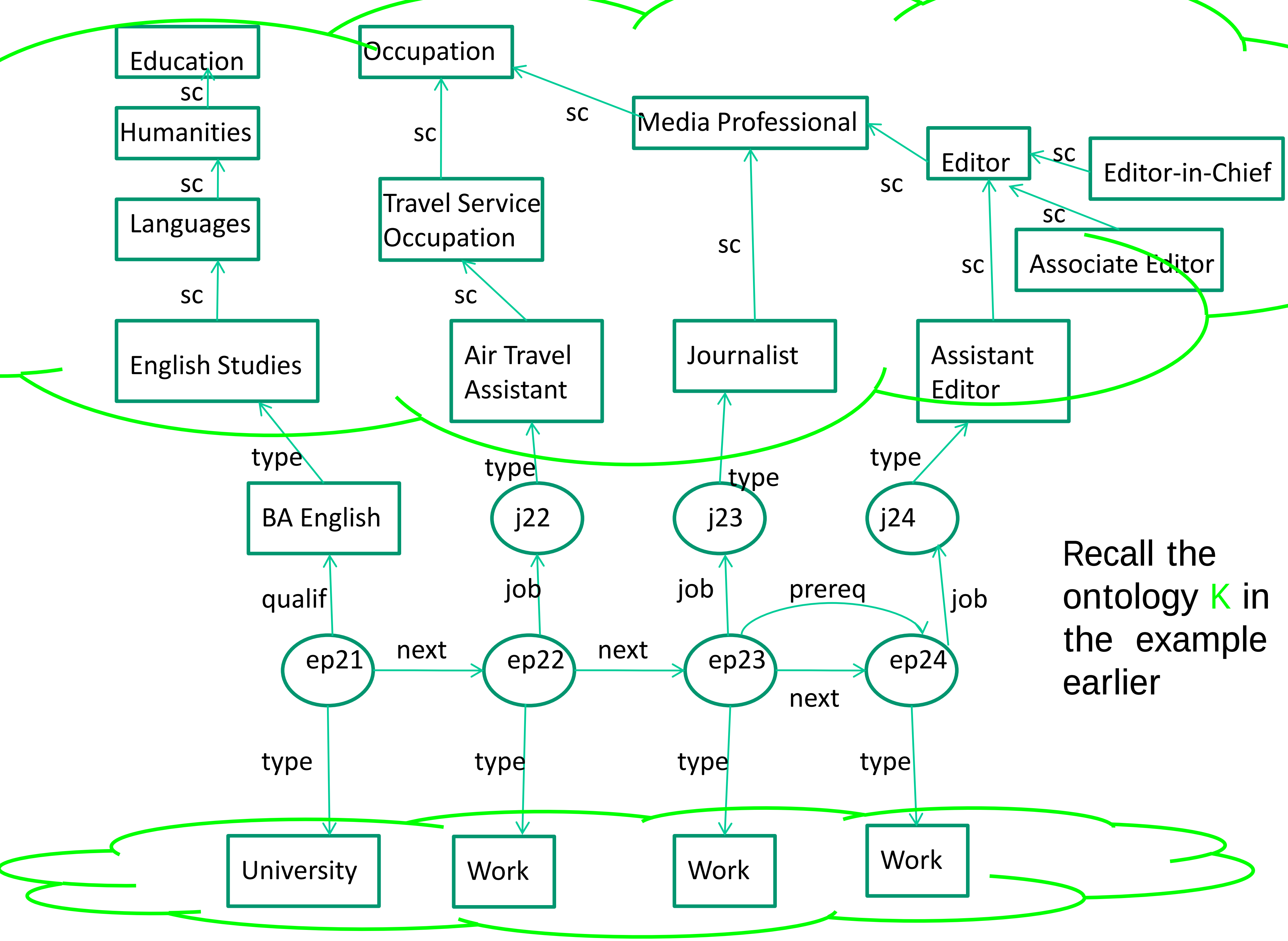
- For each query conjunct, use techniques described earlier to compute a relation r_i with scheme (X_i, Y_i, ED, RD)
- If it is an exact conjunct, then for each tuple t in r_i t[ED] = r[RD] = 0
- If it is an APPROXed conjunct, then for each tuple t in r_i t[RD] = 0 and t[ED] is the edit distance for 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:

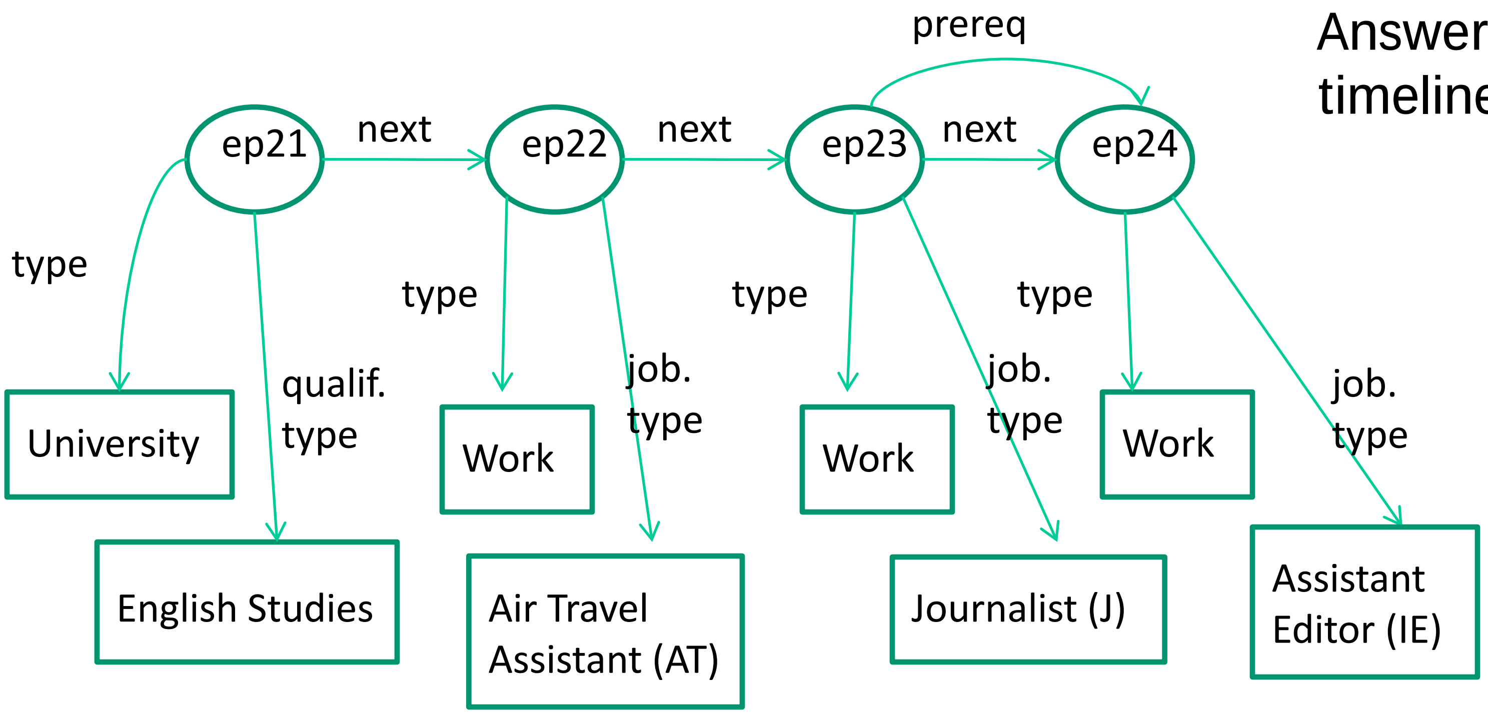
```
(?E2,?P) ← (?E1,type,University),
        RELAX(?E1,qualif.type,EnglishStudies),
        APPROX (?E1,prereq+,?E2), (?E2,job.type,?P),
        APPROX (?E2,prereq+,?Goal), (?Goal,type,Work)
        RELAX (?Goal,job.type,AssistantEditor)
```

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



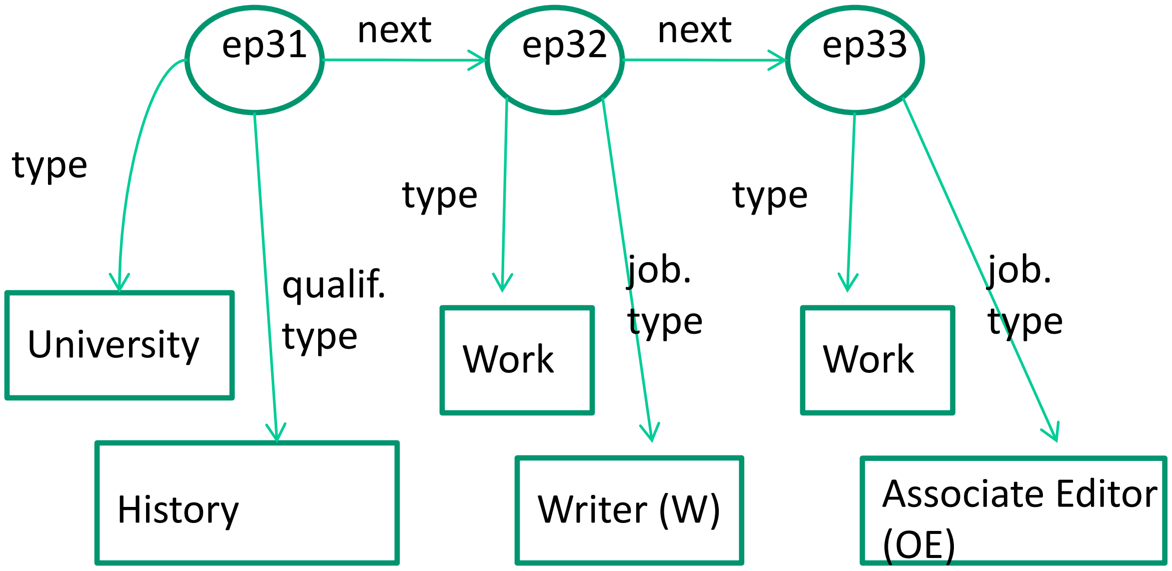
Recall the ontology K in the example earlier

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. Conclusions and Future Work

- We have presented a new technique for query relaxation of conjunctive regular path queries
- We have shown how this can be combined with query approximation to provide greater flexibility in querying of complex, irregular semi-structured data sets
- Users are able to specify approximations and relaxations to conjuncts of their query, and the relative costs of these
- Query results are returned incrementally, ranked in order of non-decreasing distance from the original query
- We have presented polynomial-time algorithms for incrementally computing the top-k answers

Future Work

- Design, prototyping and evaluation of visual query interfaces, for users to select their approximation/relaxation requirements, set relative costs, incrementally explore query results
- Empirical evaluation of our query processing algorithms, in domains such as e-learning and the life sciences, and investigation of optimisation techniques
- Merging our APPROX and RELAX operations into one integrated FLEX operation
    - this applies concurrently both approximation and relaxation to a regular path query
    - building on our common NFA-based approach to APPROX and RELAX