Flexible Querying of Lifelong Learner Metadata

Alexandra Poulovassilis, Petra Selmer, and Peter T. Wood

Abstract—This paper discusses the provision of flexible querying facilities over 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 by seeing what others have done. We motivate and describe the development of a prototype system, called ApproxRelax, that provides users with a graphical facility for incrementally constructing their queries and that supports both query approximation and query relaxation, thus allowing for flexible matching of users’ queries against the data provided by other learners. We show how the system is able to return results in ranked order of their “distance” from the user’s query. Our approach is novel both in its aim of supporting lifelong learners in reflecting on their learning and career choices, and also in its technical foundations that combine for the first time query approximation and query relaxation techniques for querying semistructured data.

Index Terms—Lifelong learning, careers guidance, semantic web, flexible querying, timelines.

1 INTRODUCTION

Supporting the needs of lifelong learners has led to research into learner centered models of delivering learning resources and opportunities [13, 12] and into the role of online support in providing careers guidance [3]. In this direction, the L4All system aims to support lifelong learners in exploring learning opportunities and in planning and reflecting on their learning [4], [21]. 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. Also, it provides a tool to understand social as well as educational factors that may influence career decisions and educational choices. In L4All, users’ timelines are stored in the form of RDF/S, through the Jena framework, as is information about courses. Users can choose to make their timelines “public” and thus accessible by other users (within “public” timelines, individual episodes may however be marked as being “private,” for example, episodes of a personal nature that the user does not wish to share with other people). This sharing of timelines exposes future learning and work possibilities that may otherwise not have been considered, positioning successful learners as role models to inspire confidence and a sense of opportunity. The system’s interface provides screens for the user to enter their personal details, as well as access to an "balloon" to pop up containing more detailed information viewed by clicking on the block representing it, causing a horizontal block representing its duration. Details of an episode can be added and deleted by users, extending or narrowing their timelines around this. Each episode is displayed in chronological order, with the system functionalities are organized as an interface. At its center is a visual representation of the user’s episode, with a primary and possibly a secondary classification. Some types of episodes are annotated by the user, when they create the system, each belonging to one of four categories: Educational, Occupational, Personal, and Other. These classifications are drawn from standard United Kingdom occupational and educational taxonomies. The system provides functionality to search over the timelines of other users, based on a variety of search criteria. However, the final evaluation of the L4All system [21] concluded that further work is needed from a technological perspective before it can be offered as an institutional service, particularly as relating to the timeline search. Specifically, problems were identified with the way in which users specify their search queries and with the ranking of the search results—we discuss these problems in Section 2. In this paper, we describe an alternative approach to supporting users’ search over timeline data, based on query approximation and query relaxation techniques.

We begin with an overview of the L4All system in Section 2, to the level of detail necessary for this paper, and discuss its key features and limitations. Motivated by this discussion, we present in Section 3 a prototype system, called ApproxRelax, which allows users to construct their queries more precisely than with L4All, and which supports flexible matching of queries, returning answers ranked in order of their “distance” from the user’s query. In Section 4, we compare this new approach with L4All. In Section 5, we discuss related work. We give our conclusions and discuss areas for further work in Section 6.

2 OVERVIEW OF L4ALL

Fig. 1 (from [21]) shows the main screen of the L4All user interface. At its center is a visual representation of the user’s timeline, and the system functionalities are organized around this. Each episode is displayed in chronological order, and is represented by an icon specific to its type (work, university, school, travel, etc.) and by a horizontal block representing its duration. Details of an episode can be viewed by clicking on the block representing it, causing a “balloon” to pop up containing more detailed information about the episode (dates, description), as well as access to edit and deletion functions.

There are some 20 types of episode supported by the system, each belonging to one of four categories: Educational, Occupational, Personal, and Other. Some types of episode are annotated by the user, when they create the episode, with a primary and possibly a secondary classification. These classifications are drawn from standard United Kingdom occupational and educational taxonomies. In particular, all Educational episodes are classified by a subject from the Labour Force Survey Subject of Degree

cter.

A. Poulovassilis and P. Selmer are with the London Knowledge Lab, Birkbeck, University of London, 23-29 Emerald Street, WC1N 3QS London, United Kingdom. E-mail: {ap, lselm01}@dcs.bbk.ac.uk.

P.T. Wood is with the Department of Computer Science and Information Systems, Birkbeck, University of London, Malet Street, WC1E 7HX London, United Kingdom. E-mail: ptw@dcs.bbk.ac.uk.

Manuscript received 1 Apr. 2011; revised 28 Nov. 2011; accepted 6 Dec. 2011; published online 12 Dec. 2011.

For information on obtaining reprints of this article, please send e-mail to: lb@computer.org, and reference IEEECS Log Number TLTSI-2011-04-0041.

Digital Object Identifier no. 10.1109/TLT.2011.38.
(SBJ) classification and a qualification level from the National Qualifications Framework (NQF). All work and voluntary Occupational episodes are classified by an industry sector from the Standard Industrial Classification (SIC) and an occupation/position from the Standard Occupational Classification (SOC). We refer the reader to the Labour Force Survey User Guide for details of these standards.\(^{1}\) In the L4All system, each classification hierarchy is limited to be up to four levels deep, so each classification annotation consists of between 1 and 4 identifiers, depending on the depth of the selected concept within the hierarchy.

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 with a similar background have gone on to do. In particular, van Labeke et al. [22], [20] describe a facility that is provided by the system for searching for “people like me.” This facility allows the user to specify which parts of their own timeline should be matched with other users’ (public) timelines, by selecting which types of episodes should be matched. The user also selects the similarity metric that should be applied and the “depth” of episode classification that should be taken into account when their episode data are being matched with others (i.e., whether 0, 1, 2, 3, or 4 of the identifiers comprising each classification annotation should be taken into account). The similarity metric can be one of: Jaccard Similarity, Dice Similarity, Euclidean Distance, and Needleman-Wunsch Distance,\(^{2}\) each described in nontechnical terms for the user. In order for the system to be able apply these similarity metrics, the users’ timelines are encoded as token-based strings. In particular, each episode is encoded as a single-token comprising a 2-letter unique identifier denoting the category of the episode, followed by up to two 4-digit codes classifying the episode according to the four levels of the taxonomies relevant for this type of episode (which may be 0, 1, or 2 taxonomies). The information about episodes’ start and end dates is ignored and only the relative position of episodes is captured. Filters are applied to the string of tokens to remove those types of episode that should not be considered in the current search, and for limiting the depth of their classification to be considered in the matching process. We refer the reader to [22] for more details of the timeline encoding and for a detailed comparison of the different similarity metrics considered for incorporation within the system.

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 normalized similarity. The user can then select one of these timelines to visualize in detail. This timeline is then displayed within the main interface screen as an extra strip below the user’s own timeline. Episodes within the selected timeline that have been designated as “public” by its owner are visible, and the user can click on any of these to expose its details and to explore it further.

An evaluation of this search for “people like me” functionality was undertaken with a group of learners at Birkbeck College (which specializes in providing flexible learning opportunities for mature students) [20]. Although they could appreciate the potential of this functionality, participants reported difficulties in understanding the meaning of some of the search parameters, namely the “depth” of episode classification and the choice of search method (similarity metric). They also identified the need for a more contextualized usage of timeline similarity matching, which explicitly identifies possible future learning and professional possibilities for the user. In follow-on work, Van Labeke et al. [21] explored a more contextualized usage of timeline similarity matching which uses just one similarity metric (hence removing this element of choice, and potential difficulty, for the user) and which explicitly shows the episodes of the selected timeline that have no match within the user’s timeline and thus represent episodes the user may be inspired to explore further for their own learning and career development. We briefly describe this “what next” facility below, in order to motivate our own approach described in Section 3.

### 2.1 What Next in L4All

The L4All “What Next” facility uses the Needleman-Wunsch similarity metric. This is because this is the only one of the metrics explored for adoption within the system that takes into consideration the positions of the tokens within the timeline encodings and hence that is able to generate an alignment between the tokens in the two timelines being matched (see [22]). Using the Needleman-Wunsch similarity metric, the What Next facility considers the distance between two strings of tokens \(x\) and \(y\) to be the minimum cost of transforming \(x\) to \(y\) by a series of insert or delete operations (replacement operations are not considered). The system builds a cost matrix incrementally by constructing a cost value for each pair of tokens \(x_i\) and \(y_j\) from each string, as follows:

\[
\text{cost}(x_i, y_j) = \begin{cases} 
\text{cost}(x_{i-1}, y_{j-1}), & \text{if } x_i = y_j, \\
\min(\text{cost}(x_{i-1}, y_j) + G, \\
\text{cost}(x_i, y_{j-1}) + G), & \text{otherwise},
\end{cases}
\]

where \(G\) is the cost of a “gap” in one of the strings (set to 1 in practice). The final cost is the cost at the matrix entry

2. See [www.dcs.shef.ac.uk/~sam/stringmetrics.html](http://www.dcs.shef.ac.uk/~sam/stringmetrics.html).
where \((x_n, y_m)\), where \(n\) is the number of tokens in \(x\) and \(m\) the number of tokens in \(y\).

A summary of the relevant timelines found by the system is presented to the user, ordered by their similarity to the user’s timeline with respect to the specified parameters and summarized by a short description. The user can now select one of these timelines and see it displayed in the main window, below their own timeline—see Fig. 2 (from [21]). The information arising from the token-based alignment between the two timelines is used to indicate, using different colors, the status of each episode in the selected timeline.

- Blue is used for episodes that match episodes in the user’s own timeline—what is termed the “common ground” between the two timelines; in Fig. 2, these are the episodes “Museum Curator” and “Foundation Degree (FD) IT.”
- Orange is used for episodes in the target timeline that occur after all blue episodes; these are deemed by the system to be relevant as a potential source of inspiration for this user as they occur after the matching episodes, and thus represent episodes the user may be inspired to explore further for their future learning and career development; in Fig. 2, these are the episodes “System Network Engineer,” “System Support Engineer,” “Data Center Support Team Leader.”
- Gray is used for episodes deemed to be irrelevant—these are episodes in the target timeline that occur earlier than all blue episodes and will mostly include earlier experiences that will be irrelevant for the user; in Fig. 2, this is the episode “A Level”; gray is also used for episodes that occur interspersed among blue episodes in the target timeline, but have no match with episodes in the user’s own timeline; in Fig. 2, these are the episodes “Diploma in CS” and “Courier.”

Van Labeke et al. [21] report on the results of two evaluation sessions that were held with mature learners at Birkbeck College and at the College of North East London assessing the L4All system. Overall, there was satisfaction with the main functionalities provided the system. Participants could see its potential value in helping learners reflect on their current learning, gaining self-confidence, and identifying possibilities for their future work and learning episodes. However, the evaluation sessions used as their sample timelines those of two current students on a Foundation Degree in IT and those of five recent alumni from the same course. Evaluation of the What Next functionality involved each participant logging in as one of the current students and searching over the other timelines, which by definition provided useful matches with the user’s timeline (since the FD course episode which was the last episode in the user’s timeline matched the same FD course episode within the alumni’s timelines). Van Labeke et al. [21] highlight that further work is needed from a technological perspective before L4All can be offered as an institutional service. In particular, they identify three issues that need to be explored more deeply.

- The top-ranked timelines in the list returned by What Next will be timelines that are most similar to the user’s own timeline. These timelines may in practice offer few suggestions of episodes for the user’s future development. The study in [21] avoided this problem by using a small set of timelines from current students and a larger set of timelines of alumni from the same course.
- What level of detail should be used in episode classification for the purpose of applying the similarity metric: selection of different classification levels by the user will give rise to different similarity values, and therefore different possible timeline alignments.
- Using the distance matrix computed by the Needleman-Wunsch algorithm to generate the episode alignments may generate several possible alignments between two timelines. Determining the “best” one in a given context is not easy, as subjective factors relating to the user’s own definition of “relevance” have to be taken into account. The current implementation in L4All always makes the same default choice of alignment, whereby the “common ground” of matching episodes in the two timelines is selected to be as late as possible within the user’s timeline.

The above problems arise because the What Next facility is rather rigid: it uses the whole of the user’s timeline; it offers just one similarity metric over the timeline data; it allows just a single level of detail to be applied to the classifications of the selected categories of episode for the similarity matching; and the similarity matching is applied...
to all episodes of these categories in the user’s timeline. 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.

3 FLEXIBLE QUERY PROCESSING

We now present a prototype system—called ApproxRelax—which supports more flexible user querying of timeline data.

The data model underlying the ApproxRelax prototype is a semistructured one, comprising a directed graph \( G = (V, E) \) and an ontology \( O = (V_o, E_o) \). \( V \) contains nodes representing an entity instance or an entity class. \( E \) contains edges representing relationships between members of \( V \). Each node in \( V \) is labeled with a distinct constant. Each edge in \( E \) is labeled with the name of a relationship, drawn from a finite set of labels \( \Sigma \), or with the label type. \( V_o \) contains nodes representing an entity class or a property. Each node in \( V_o \) is labeled with a distinct constant. The subset of nodes in \( V \) that represent entity classes is contained in \( V_o \). The set of labels of edges in \( E \), except for the label type, is contained in the set of labels of property nodes in \( V_o \). Each edge in \( E_o \) is labeled with one of: \( sc \), \( sp \), \( dom \), or \( range \). We assume that \( \Sigma \cap \{ \text{type}, \text{sc}, \text{sp}, \text{dom}, \text{range} \} = \emptyset \). We note that this model encompasses RDF data, except that it does not allow for the representation of RDF’s “blank” nodes. It also encompasses a fragment of the RDFS vocabulary: \( \text{rdfs:} \text{type}, \text{rdfs:subClassOf}, \text{rdfs:subPropertyOf}, \text{rdfs:domain}, \text{and} \text{rdfs:range} \), abbreviated here by \( \text{type}, \text{sc}, \text{sp}, \text{dom}, \text{and} \text{range} \), respectively.

Fig. 3 illustrates a fragment of data and metadata relating to a user’s timeline (Dan). The oval nodes and the nodes adjacent to them are contained in \( V \). The edges between these nodes are contained in \( E \). The rectangular nodes are contained in \( V_o \) (class nodes). The labels of edges in \( E \) (except type) are also contained in \( V_o \) (property nodes). \( E_o \) contains the edges labeled \( sc \) between class nodes, and also edges labeled \( dom \) and \( range \) linking each property node to a class node (not shown in the figure). There are several types of episode, e.g., UniversityEpisode and WorkEpisode. Associated with each type of episode are several properties, e.g., qualification and job. Episodes are ordered according to their start date—as indicated by edges labeled next (for simplicity, the episodes’ start and end dates are not shown). If two episodes have the same start date, the one that ends earlier is considered to precede the other. If two episodes have identical start and end dates, an arbitrary one is chosen as being the earlier one. An edge labeled \text{prereq} from one episode to another is an annotation created by the timeline’s owner indicating that they consider that undertaking an earlier episode was necessary in order for them to be able to proceed to or achieve a later episode.

The query language underlying the ApproxRelax prototype is that of Conjunctive Regular Path (CRP) queries [2]. A conjunctive regular path query, \( Q \), consisting of \( n \) conjuncts is 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 \( Y_i \) is a variable or a constant, each \( Z_i \) is a variable appearing in the right-hand side of \( Q \), and each \( R_i \) is a regular expression over the alphabet from which edge labels in the graph are drawn. In our context, a regular expression \( R \) is defined as follows:

\[
R := \epsilon | a | (R_1 \cdot R_2) | (R_1|R_2) | R^* | R^+,
\]

where \( \epsilon \) is the empty string, \( a \) is any symbol in \( \Sigma \cup \{ \text{type} \} \), “\( \cdot \)” denotes the disjunction of all constants in \( \Sigma \cup \{ \text{type} \} \), and the operators have their usual meaning.

The (exact) answer to a CRP query \( Q \) on a graph \( G \) can be obtained as follows: we first find, for each \( 1 \leq i \leq n \), 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 \( G \) that satisfies \( R_i \), that is, whose concatenation of edge labels is in \( L(R_i) \) (the language recognized by the regular expression \( R_i \)). We then form the natural join of relations \( r_1, \ldots, r_n \) and project over \( Z_1 \) to \( Z_m \).

**Example 1.** Suppose Gaby is studying on the Foundation Degree in Information Technology (FdIT) at Birkbeck College and she wishes to find out what possible future career choices there may be for her by seeing what other people with qualifications in Information Systems have gone on to do in their careers. This can be undertaken by the following CRP query, \( Q_1 \), over the timeline data (we stress that this is not what a user would enter; in the actual ApproxRelax system, queries such as this are automatically generated through users’ interactions with the system’s graphical user interface, which we describe in Section 3.1):

\[
(?A, ?B, ?C) \leftarrow
(?A, \text{type,UniversityEpisode}),
(?A, \text{qualif.type,InformationSystems}),
(?A, \text{prereq,?B}),
(?B, \text{type,WorkEpisode}),
(?B, \text{job.type,?C})
\]

(Variables in a query have an initial question mark mark.) However, \( Q_1 \) returns no results relating to Dan’s timeline in Fig. 3, even though this timeline contains information that could be relevant to Gaby (because Dan studied Information Systems at the university and then undertook several IT-related work episodes). No results are
returned because the above query specifies prereq as the edge connecting successive episodes, whereas users may or may not create full prereq metadata relating to their episodes. Although ?A can be instantiated to episode dan1 from Dan’s timeline to satisfy the first two query conjuncts, there is no edge prereq from dan1 to satisfy the third conjunct.

In [9], we investigated approximate matching of CRP queries, allowing edit operations such as insertions, deletions, and substitutions of edge labels to be applied to the regular expressions $R_i$, each with some edit cost. The ApproxRelax prototype currently supports insertions and deletions, each with a cost of $\alpha$. The user can configure the system to apply either insertion or deletion edit operations, or both, and can also set the value of $\alpha$.

**Example 2.** Recall Example 1 where Gaby’s original query returned no answers because users may not create full prereq metadata relating to their timelines. To allow for such irregularities in the timeline data, Gaby can instead submit a variant of $Q_1$ in which the conjunct (?A, prereq, ?B) can be approximated:

\[
(?A, ?B, ?C) \leftarrow \\
(?A, \text{type}, \text{UniversityEpisode}), \\
(?A, \text{qualif.type}, \text{InformationSystems}), \\
\operatorname{APPROX}(?A, \text{prereq}, ?B), \\
(?B, \text{type}, \text{WorkEpisode}), \\
(?B, \text{job.type}, ?C)
\]

Assuming that insertion and deletion edit operations are applied, the regular expression prereq can be approximated by regular expressions $\epsilon$ and next.prereq, both at edit distance $\alpha$ from prereq (among other approximations). There are still no answers from Dan’s timeline returned at edit distance $\alpha$. Further approximation of the regular expression $\epsilon$ to next does allow an answer to be returned from Dan’s timeline at edit distance $2\alpha$, namely (dan1, dan2, IT Operations Technicians). Approximation of the regular expression next.prereq to next.next.prereq also allows another answer to be returned at edit distance $2\alpha$, namely (dan1, dan4, IT User Support Technicians). Gaby may judge either of these results to be relevant to her and can now ask the system to return the whole of Dan’s timeline for her to visualize and explore further.

In [16], we investigated allowing also ontology-based relaxation to be applied to the regular expressions $R_i$. This encompasses query relaxations that are entailed using information from the ontology $O$, such as replacing a class label by that of a superclass or a property label by that of a superproperty. The ApproxRelax prototype implements, for the first time, ontology-based relaxation of regular path queries.

**Example 3.** Suppose that Gaby decides that she is interested in jobs categorized under Software Professionals and similar categories. She also decides to broaden her search by allowing qualifications that are similar to Information Systems. She might submit the following query, $Q_3$:

\[
(?A, ?B) \leftarrow \\
(?A, \text{type}, \text{UniversityEpisode}), \\
\operatorname{RELAX}(?A, \text{qualif.type}, \text{InformationSystems}), \\
\operatorname{APPROX}(?A, \text{next}, ?B), \\
(?B, \text{type}, \text{WorkEpisode}), \\
\operatorname{RELAX}(?B, \text{job.type}, \text{SoftwareProfessionals})
\]

This query allows Information Systems to be relaxed to its parent concept Mathematical & Computer Sciences, matching qualifications such as Computer Science, etc. (see also the timelines in Figs. 4 and 5); and Software Professionals to be relaxed to its parent concept Information & Communication Sciences.
Technology Professionals, matching occupations such as IT Strategy & Planning Professionals, etc. In parallel with this query relaxation, the third query conjunct is also being approximated at the same time.

Query answers will be returned to Gaby in increasing overall distance from the nonapproximated, non-relaxed version of her query, $Q_2^{EXACT}$. Query $Q_2^{EXACT}$ is the same as query $Q_2$ but without the APPROX and the two instances of RELAX. The overall distance of a query answer from $Q_2^{EXACT}$ is the sum of the costs of the relaxation and edit operations that were required to be applied to $Q_2^{EXACT}$ in order to find that answer. We will return to the evaluation of $Q_2$ in more detail in Sections 3.2-3.4.

3.1 The ApproxRelax Prototype

The ApproxRelax prototype provides users with a graphical user interface through which they can formulate their queries. We illustrate the process by considering how Gaby can formulate query $Q_2$. In order to facilitate ease of use, the more complex parts of the user interface are explained to the user by means of “tooltips” and hover-over text (indicated by boxes containing a “?”). ApproxRelax is a web application and to use it Gaby opens up her browser and proceeds to the screen shown in Fig. 6. This screen allows the user to start formulating a query by creating a query template for matching educational episodes or occupational episodes.

Gaby clicks the “Create an educational episode” image and is presented with the screen shown in Fig. 7. From the “Type” drop-down menu she is able to make a choice from the Educational episode types, and she selects “University Episode.” From the “Subject” drop-down menu she is able to make a choice from different subject areas (as sourced from the SBJ taxonomy mentioned in Section 2). She selects “Information Systems” and ticks the “Fetch similar or related subjects?” checkbox.

As she has not yet finished constructing her query, she clicks the “Next” button. At this point, the system generates internally these query conjuncts:

$$(?A, type, UniversityEpisode)$$
RELAX($$?A, qualif.type, InformationSystems)$$

The first of these is generated from Gaby’s selection of “University Episode” in the “Type” drop-down. The fact that Gaby designated this episode as an educational episode and selected “Information Systems” from the “Subject” drop-down gives rise to the second conjunct. Because she ticked the “Fetch similar or related subjects?” checkbox, this conjunct additionally has the RELAX keyword applied to it by the system.

Having clicked “Next,” Gaby is presented again with the screen in Fig. 6. Gaby now clicks the “Create an occupational episode” image and is presented with the screen shown in Fig. 8. As this is not the first episode of the query, there is a “Link from previous episode” drop-down, which allows the user to specify the way in which the previously specified episode is related to the one currently being specified. The possible choices here (in the current prototype) are next, next+, prereq, and prereq+, which are displayed in the drop-down using more user-friendly descriptions: “next episode,” “next or subsequent episode,” “direct prerequisite,” and “direct or indirect prerequisite.”

Gaby selects the “next episode” option and ticks the “Flexible matching of the link between this episode and the previous one?” checkbox. From the “Type” drop-down menu she is able to make a choice from the Occupational episode types, and she selects “Work Episode.” From the “Job” drop-down menu she is able to make a choice from different jobs (as sourced from the SOC taxonomy mentioned in Section 2). Gaby selects “Software Professionals” and ticks the “Fetch
similar or related occupations?” checkbox. She has now finished constructing her query and clicks the “Done” button. At this point the system generates internally the following query conjuncts:

\[ \text{APPROX}(?A, \text{next}, ?B) \quad \text{[C3]} \]
\[ (?B, \text{type}, \text{Work Episode}) \quad \text{[C4]} \]
\[ \text{RELAX}(?B, \text{job.type, SoftwareProfessionals}) \quad \text{[C5]} \]

Conjunct C3 links the query episode set up previously (denoted by ?A) to this second episode (denoted by ?B). It contains the selection made by Gaby in the “Link from previous episode” drop-down. Additionally, as Gaby has ticked the “Flexible matching …” checkbox, C3 has the APPROX keyword applied to it. Conjunct C4 is generated from Gaby’s selection of “Work Episode” in the “Type” drop-down. The fact that Gaby designated this episode as an occupational episode, and selected “Software Professionals” from the “Job” drop-down gives rise to conjunct C5. Since Gaby ticked the “Fetch similar or related occupations?” checkbox, C5 additionally has the RELAX keyword applied it. The system provides the facility for the user to view the details of previously constructed query templates, as may be seen on the right-hand side of Fig. 8; more about this facility is detailed in the next paragraph.

The next screen that Gaby is presented with is shown in Fig. 9. It allows the user to view at a glance the query templates making up their query (this is identical to the facility seen on the right-hand side of Fig. 8) and allows the user to view previously constructed query templates while in the process of creating their query. The type of each episode is immediately clear, as denoted by the image. In Fig. 9, the second image has been clicked (its number is highlighted in red) and the information pertaining to Gaby’s second query template is displayed. Gaby can now click on the “cog” image (which has the relevant tooltip) to execute her query.

Gaby is now presented with the screen shown in Fig. 10, which displays query results ranked in order of increasing distance from the nonapproximated, nonrelaxed version of her query. The derivation of these query results is discussed in more detail in the sections that follow. For each result, an avatar representing the timeline’s owner is displayed, as well as their name, the episode in their timeline which matches the last query template of the user’s query, the distance at which this result has been retrieved, and an automatically generated summary of the timeline’s owner and contents of their timeline. The latter description is displayed to give the user an overview of the matching timeline so that they can decide whether they wish to explore it in more detail.

At present, this is as far as the ApproxRelax prototype goes in terms of displaying query results. As part of future work, we intend that clicking on the timeline owner’s name will take the user to a screen similar to the “What Next” visualization in Fig. 2 earlier. The selected timeline would be displayed in the bottom part of this new screen. The top part of the screen would display a visual representation of the user’s query, showing one block for each of the query templates, and aligning each block above the episode that it matches in the timeline displayed below. Also missing from the current ApproxRelax prototype are abilities for querying additional classifications according to the NQF and SIC taxonomies mentioned in Section 2, and for formulating query templates for Personal and Other episode types.

3.2 Query Evaluation without Approximation or Relaxation

To see how queries such as \( Q_2 \) above are evaluated in the ApproxRelax prototype, we first consider the evaluation of exact CRP queries, without any approximation or relaxation, first comprising just one conjunct and then comprising multiple conjuncts.

A single-conjunct CRP query, \( Q \), over a graph \( G = (V, E) \) is of the form
where $X$ and $Y$ are constants or variables, $R$ is a regular expression over $\Sigma \cup \{\text{type}\}$ as defined earlier, and $vars$ is the subset of $\{X, Y\}$ that are variables.

In order to compute the answer to a single-conjunct CRP query, we first construct a weighted NFA $M_R$ to recognize $L(R)$, using Thompson’s construction (which makes use of $\epsilon$-transitions) \cite{1}. Each transition of $M_R$ is labeled with a label from $\Sigma \cup \{\text{type}\}$ and has a weight (i.e., cost) which is zero. If $X$ in the query is a constant $m$, we annotate the initial state, $s_0$, of $M_R$ with $m$; otherwise we annotate $s_0$ with a wildcard symbol “*” that matches any constant. Likewise, depending on whether $Y$ in the query is a variable or a constant $m$, the final state, $s_f$, of $M_R$ is annotated with “*” or $m$.

We next form the weighted product automaton, $H$, of $M_R$ with the graph $G = (V, E)$, viewing each node of $V$ as both an initial and a final state. The states of $H$ are of the form $(s, n)$, with $s$ being a state of $M_R$ and $n \in V$. There is a transition in $H$ from state $(s, u)$ to state $(t, v)$ with label $l$ and weight $w$ if and only if there is a transition with label $l$ and weight $w$ from $s$ to $t$ in $M_R$ and an edge labeled $l$ from $u$ to $v$ in $G$. (Label $l$ can be $\epsilon$, in which case $u = v$ in $G$.)

To evaluate query $Q$ above, if $X$ is a node $m$ of $G$, we perform a shortest path traversal of $H$ starting from the state $(s_0, m)$. Whenever we reach a state $(s_f, n)$ of $H$, we output $n$ provided that $n$ matches the annotation on $s_f$. If $X$ is a variable, we perform such a traversal of $H$ starting from the state $(s_0, v)$ for every node $v$ of $G$. All exact answers returned always have a cost of zero, of course.

Turning now to the evaluation of multiconjunct CRP queries, a query evaluation tree is constructed for such a query, consisting of inner nodes denoting join operators and leaf nodes representing individual query conjuncts. The query is evaluated by joining the answers arising from the evaluation of each of its conjuncts, each of which is computed as the query tree is traversed.

Example 4. Consider query $Q_2$ from Example 3 earlier, and in particular a variant, $Q_2^{\text{EXACT}}$, containing no approximation or relaxation operations:

\[
(?A, ?B) \leftarrow \\
(?A, \text{type}, \text{University Episode}), \quad [C1] \\
(?A, \text{qualif. type}, \text{InformationSystems}), \quad [C2] \\
(?A, \text{next}, ?B), \quad [C3] \\
(?B, \text{type}, \text{WorkEpisode}), \quad [C4] \\
(?B, \text{job.type}, \text{SoftwareProfessionals}) \quad [C5]
\]

The answers produced for $Q_2^{\text{EXACT}}$ are shown in Fig. 11.

The five columns refer to the answers produced for each of the five conjuncts of $Q_2^{\text{EXACT}}$, showing in each case the instantiations of the conjunct’s variables. The tuples that contribute to the final answer are displayed in bold. We see that the only answer for $Q_2^{\text{EXACT}}$ is (liz1, liz2).

3.3 Query Evaluation with Approximation

We now consider the evaluation of queries in which conjuncts may be prefixed with the APPROX keyword. Consider such a single-conjunct CRP query, $Q$

\[
vars \leftarrow \text{APPROX}(X, R, Y).
\]

The edit operations that are currently supported in the ApproxRelax prototype are insertions and deletions of edge labels, each with an edit cost of $\alpha$ (whose value can be configured by the user). The edit distance from a path $p$ in graph $G = (V, E)$ 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 distance of a path $p$ to a regular expression $R$ is the minimum edit distance from $p$ to any path that conforms to $R$. Given a matching $\theta$ from the variables and constants of query $Q$ to nodes in $G$, such that constants are matched to themselves, we say that the tuple $\theta(vars)$ has edit distance $\text{edist}(\theta, Q)$ to $Q$, defined as the minimum edit distance to $R$ of any path $p$ from $\theta(X)$ to $\theta(Y)$ in $G$. Note that if $p$ conforms to $R$, then $\theta(vars)$ has edit distance zero to $Q$—this is the exact matching case, as described in Section 3.2.

The approximate answer of $Q$ on $G$ is a list of pairs $(\theta(vars), \text{edist}(\theta, Q))$, ranked in order of nondecreasing edit distance. The approximate top-$k$ answer of $Q$ on $G$ comprises the first $k$ tuples in the approximate answer of $Q$ on $G$.

In order to compute the answer to an approximated single-conjunct CRP query, we construct the approximate automaton $A_R$ corresponding to the automaton $M_R$, whose construction was described in Section 3.2. $A_R$ augments $M_R$ with additional transitions that capture the insertion and deletion edit operations. For insertions, each state $s$ in $M_R$ is augmented with $|\Sigma|$ transitions, one for each $a \in \Sigma$, from $s$ back to $s$. For deletions, each transition from a state $s$ to a state $t$ in $M_R$ labeled with a symbol $a \in \Sigma$ gives rise to a transition in $A_R$ from $s$ to $t$ labeled with $\epsilon$ (the empty string). Each new transition in $A_R$ has a cost of $\alpha$. We form again the weighted product automaton, $H$, but this time of $A_R$ with $G$. In the same manner as previously described, we perform one or several shortest path traversals of $H$ (depending on whether $X$ is a constant) and upon reaching a state $(s_f, n)$ we output $n$ provided that it matches the annotation on $s_f$.

In contrast to Section 3.2, the answers retrieved so far are now stored in a list answers, ordered by nondecreasing edit distance. A new answer is added to this list only if it is not already in the list—this is to avoid returning the same answer multiple times, at increasing distances from the original query. Answers which are an exact match will be returned at a distance of zero. Any approximate answer will be returned at some nonzero cost equal to the edit distance (which will be some multiple of $\alpha$).

To illustrate this process, consider the single-conjunct query $\text{APPROX(liz1, next, ?B)}$ which refers to the timeline in Fig. 4, and suppose that $\alpha = 1$. We see that the only answer for this conjunct at distance 0 is liz2. There is an
Consider a relaxed single-conjunct CRP query, $Q^\text{APPROX}_2$ of variables and constants of $Q$ to nodes in the graph $G$, such that constants are matched to themselves. We denote $(\theta(X), R, \theta(Y'))$ by $\theta(Q)$. We represent paths $p$ in $G$ and strings $q \in L(R)$ in the form of a set of triples comprising a source and target node and an edge label (see [16]). A path $p$ in $G$ $r$-conforms to $\theta(Q)$ if there is a string $q \in L(R)$ such that the triple form of $q$ relaxes to the triple form of $p$. The relaxation distance from $p$ to $\theta(Q), q$ is the minimum cost of any sequence of direct relaxations which yields the triple form of $p$ from that of $q$. The relaxation distance from $p$ to $\theta(Q)$ is the minimum relaxation distance from $p$ to $\theta(Q), q$ for any string $q \in L(R)$. The relaxation distance of $\theta(Q)$, denoted $\text{rdist}(\theta(Q), q)$, is the minimum relaxation distance to $\theta(Q)$ from any path $p$ that $r$-conforms to $\theta(Q)$.

The relaxed answer of $Q$ on $G$ is a list of pairs $(\theta(\text{vars}), \text{rdist}(\theta(Q)))$ ranked in order of nondecreasing relaxation distance. The relaxed top-$k$ answer of $Q$ on $G$ comprises the first $k$ tuples in the relaxed answer of $Q$ on $G$.

In order to compute the answer to a relaxed single-conjunct CRP query, we construct the relaxed automaton $O_R$ corresponding to $M_R$, whose construction was described in Section 3.2. For each transition $(s, \text{type}, t)$ of weight $d$ such that $t$ is a final state annotated with the constant $c$ and $c^e$ is an immediate superclass of $c$ in $O$, we add: 1) a new final state $t^r$ annotated with $c^e$, 2) a copy of all of $t$’s outgoing transitions to $t^r$, and 3) a new transition $(s, \text{type}, t^r)$ of weight $d + \beta$. We repeat this until no more states and transitions can be inferred. The process terminates because of our assumption that the subgraph of $O$ induced by edges labeled $ac$ is acyclic.

We form again the weighted product automaton, $H$, but this time of $O_R$ with $G$. In the same manner as previously described, we obtain answers by traversing $H$, and store these in a list answers, ordered by nondecreasing relaxation distance.

The evaluation of general multiconjunct CRP queries by the ApproxRelax prototype is undertaken incrementally, outputting $k$ results at a time (the value of $k$ is configurable), and it proceeds as follows:

1. A query tree is constructed, whose leaf nodes are query conjuncts and inner nodes are join operators. This is achieved by first constructing the (acyclic) hypergraph of the conjuncts [19]. The query tree supports an Iterator interface, implementing the open and getNext functions. The evaluation of the query commences by invoking the query tree’s open function, which initializes the internal structures needed for each inner node and each leaf node.
2. $M_R$-type automata are constructed for conjuncts that are not approximated or relaxed. The first $k$ exact answers for these are computed in the first iteration.
3. $A_R$-type and $O_R$-type automata are constructed for the approximated and relaxed constructs, respectively. Incremental construction of $H$ is undertaken for each conjunct, whereby its nodes and edges are computed only as far as the maximum distance.
required in order to retrieve the top \( k \) results for the current iteration. Answers are computed and stored in nondecreasing distance order.

4. Evaluation of the query commences by the invocation of \( \text{getNext} \) on the root node. \( \text{getNext} \) cascades further down the tree until a conjunct node is reached. \( \text{getNext} \) on a conjunct node computes the answers for the conjunct in order of nondecreasing distance. The query tree is then traversed in a bottom-up fashion, during which the answers from each conjunct undergo a natural join operation with the answers of their sibling conjunct node upon invocation of their parent node’s \( \text{getNext} \) function. For each pair of tuples that are joined, their individual distance values are added in order to obtain the distance value of the resulting tuple. The results from this join operation are then pipelined upward, providing input for the next level’s join operation and so on, until the root of the tree is reached.

To illustrate, we return to the evaluation of the original query \( Q_2 \) from Example 3, which contains both approximation and relaxation operations. We assume that \( \alpha \) has been set to 1 and \( \beta \) to 2. The answers produced for \( Q_2 \) are shown in Fig. 13, where the answers produced for each individual conjunct in \( Q_2 \) are shown in the first five columns. The final column shows the overall query answers, in order of nondecreasing total distance. The conjunct answer tuples that contribute to the final answer tuples are italicized and are subscripted with the ordering of the final answer tuple (i.e., subscript \( i \) denotes the \( i \)th final answer tuple). In this figure, episodes \( d_i, l_i, \) and \( a_i \) are short for \( \text{dan}_i, \text{liz}_i, \) and \( \text{al}_i, \) respectively. We see that \( Q_2 \) returns seven more answers than \( Q_{\text{APPROX}} \): \( \langle \text{al}1, \text{al}2 \rangle \) at distance 2, \( \langle \text{liz}4, \text{liz}5 \rangle \) at distance 8, \( \langle \text{dan}1, \text{dan}2 \rangle \) at distance 8, etc.

3.5 System Architecture of ApproxRelax

Fig. 14 illustrates the architecture of the ApproxRelax prototype. The prototype was developed using the Microsoft .NET 3.5 framework, and follows the MVC design architecture.

When Gaby created her episodes as shown in Figs. 7 and 8, the Episode creator module in the web user interface layer managed these requests. The SBJ and SOC data (mentioned in Section 2) are stored in the data store layer. These data are used to populate the “Subject” and “Occupation” drop-downs. Access to the data store is facilitated by the Jena bridge module in the system layer. Episode creator also manages other aspects of the episode creation process, such as marking an episode for relaxation, or a link from a previous episode for approximation. This module invokes the Conjunct builder, which creates the query conjuncts from the query templates. When Gaby ran her query as shown in Fig. 9, the Query submitter module was invoked. It made a call to the Episode creator module in order to obtain all of the query conjuncts. The Query submitter then invoked the Query manager module, passing the list of conjuncts, as well as parameters such as the values for \( \alpha \) and \( \beta \), which edit operations to apply, and how many results to return. Once the Query manager module has obtained the results, it invokes the Result manager module, which manages the display of the results in ranked order.

The system layer constitutes the bulk of the processing functionality. Query evaluation commences once Query submitter invokes Query manager with the conjuncts and aforementioned parameters. Query manager invokes the Query Tree builder module with the conjuncts, which constructs the query tree and passes it back to Query manager. This next passes the query tree to the Query Tree initialiser module, which initializes the various structures needed for query evaluation. Whenever Query Tree initialiser encounters a conjunct within the query tree, the Conjunct initialiser module is invoked on the conjunct. This invokes the NFA builder.
module to construct the automaton $M_R$ corresponding to the conjunct's regular expression $R$. If the conjunct is approximated or relaxed, NFA manager will be invoked to transform $M_R$ to $A_R$ or $O_R$, respectively.

The initialized query evaluation tree is then passed back to Query manager, whereupon Query Tree evaluator is invoked. This traverses the tree, starting from the leftmost leaf node, and proceeding upward. If the node is a leaf the ranked answers for the query conjunct are computed by Conjoint evaluator. This module forms the weighted product automaton, $H$, of the conjunct's automaton with the graph, $G$, whose nodes are sourced from the timeline data stored the data store. The construction of $H$ is computed incrementally, and only includes nodes and edges relevant to the maximum distance required in order to retrieve the next $k$ results. Conjoint evaluator traverses $H$ to obtain the ranked answers. For nodes representing joins, Query Tree evaluator works in conjunction with Join manager to perform a ranked join of the answers obtained thus far. Once the root of the query tree has been reached, the processing terminates and the list answers now holds the results, ranked by increasing distance. Query manager passes this list to the Result manager.

### 4 Comparison with What Next

A fundamental difference between L4All’s What Next and the ApproxRelax prototype is that with ApproxRelax the user can pose search queries that are different from their own timelines, e.g., where some episodes in their timeline are not included in the search query, or if included they are not approximated, or where there are episodes in the search query not related to their own timeline.

Relaxation in ApproxRelax is also more flexible than in L4All: in L4All the information about each episode is encoded as a single token and the same depth of classification is applied to all types of episodes for similarity matching, whereas in ApproxRelax each query template results in several query conjuncts each of which can be individually approximated or relaxed. As a consequence of this finer level of representation, each classification can be relaxed independently (see Example 3) and answers resulting from fewer relaxations will automatically be ranked higher by the system.

Considering the three issues identified at the end of Section 2.1, the problem of the top-ranked timelines in L4All being very similar to the user’s own timeline is avoided in ApproxRelax by not requiring that all of a user’s timeline is matched against the timeline data. For example, consider the results from What Next shown in Fig. 15 (from [21]). The table shows for each of three users (al1, al3, al4) the similarity of five other timelines, in the order that these are ranked by L4All. Also shown in the fourth column is the number of suggestions of possible future episodes offered to the user (i.e., the episodes colored orange in the selected timeline). One can see here the problem of the top-ranked timelines being very similar to the user’s timeline and offering few or no suggestions to the user. The fifth column shows how many of the suggestions offered are actually relevant for this user, as rated by a lifelong learning (LL) practitioner with domain expertise in careers in IT. This person, Pract-1, had worked with diverse groups of mature students and with other LL stakeholders in deriving the requirements of the original L4All system and in undertaking successive user evaluations of the system. Pract-1 was, therefore, intimately familiar with the target user groups for L4All and of their needs from a system that aims to support lifelong learning and career choices.

In contrast, in collaboration with two LL practitioners, we used ApproxRelax on the same set of timelines, submitting three queries relevant to each of al1, al3, al4 (i.e., nine queries in all). The two LL practitioners were Pract-1, already mentioned, and Pract-2, who is director of the Foundation Degree in IT at Birkbeck, which targets mature students wishing to enter the IT profession or advance their careers in IT. Collectively therefore, they had much expertise and experience in the lifelong learning landscape and in the needs of lifelong learners and were able to provide authoritative feedback in the evaluation of ApproxRelax. For all nine queries, the “Link from previous episode” was “next episode” and the boxes “Flexible matching of the link...” and “Flexible matching of the link...” were ticked. Only the “insert” edit operation was selected to be applied. For all1, the three queries comprised one query template representing their last educational episode and a second one representing either 1) an Information Systems university episode, or 2) an ICT Managers work episode, or 3) a Software Professionals work episode. For al3 and al4, the three queries comprised two query templates representing their last educational episode and a second one representing either 1) an Information Systems university episode, or 2) an ICT Managers work episode, and a third one as in points 1-3 above. The same LL practitioner who rated the L4All query results above also rated the relevance of the ApproxRelax query results. For each user, if we add up the number of relevant results being returned ranked 1-5 across their three queries, we obtain the summary shown in Fig. 16. We also show from which timelines these relevant results are returned.
We see that more relevant results are being returned for all users by ApproxRelax compared with What Next, e.g., user al1 now has relevant results from al2, al3, al5; user al3 from al5; and user al4 from al1, al2, al5. The top-5 relevant results returned by ApproxRelax include all of the relevant suggestions from What Next, except for one relevant suggestion for users al1 and al3 from the timeline of al4 which is returned by ApproxRelax, but not in the top-5 results.

The two LL practitioners reported that they found it “much more useful” to be able to explicitly set up a search query in ApproxRelax rather than using the built-in similarity matching of L4All based on the user’s whole timeline. They also found it helpful that ApproxRelax allows users to specify what kind of episode they are looking for, for inspiration, i.e., the episodes 1-3 above. Another advantage of ApproxRelax is a clearer causality between the users’ requirements, as articulated in their search query, and the results returned by the system.

Returning to the other issues identified at the end of Section 2.1, the problem of the user having to decide on the level of classification for episode comparisons is avoided in ApproxRelax because relaxation of episode types is performed automatically by the system, with timelines containing episodes matching the user’s query in more detail being ranked higher than those matching at higher levels of classification.

The problem of finding the “best” alignment will remain difficult. However, because each query template is represented by several query conjuncts in ApproxRelax, rather than as a single token in L4All, similarity is more finely measured.

5 RELATED WORK

Research into life-course choices has highlighted two issues that contribute to lack of participation in HE: a lack of information about educational opportunities, and a perception that HE is “not for me” [23]. Social factors influence educational choices and career decisions (location, family, friends), and word-of-mouth is important in recommending educational choices [4]. Learners who receive more personalized and better targeted information may make a more successful entry to FE and HE [17]. The L4All system aims to allow potential learning and career pathways to be identified, exposing possibilities that learners may otherwise not have considered [4]. Here, we have shown how supporting query approximation and relaxation can provide greater flexibility in users’ querying of timeline data than L4All’s similarity metrics-based approach, and can provide more relevant answers.

In approximate querying, Kanza and Sagiv [10] considered querying semistructured data using flexible matchings in which the matched paths contain the labels in the query. Grahne and Thomo [6] used weighted regular transducers to transform regular path queries to match semistructured data approximately. Mandreoli et al. [15] allow edges in a query to match paths in a graph that have been semantically related (e.g., using RDFS). Approximate graph matching has also been extensively studied recently, e.g., [18].

There has been work on relaxing XML tree pattern queries, recently in [14]. Relaxation of conjunctive queries on RDF is considered in [5] and [8]. Huang et al. [7] develop a similarity measure for relaxed queries to improve the relevance of answers. Similarity-based querying was also the focus of iSPARQL [11].

In contrast to the above, the work in [16] combines within one framework both query approximation and query relaxation, and applies it to the more general query language of conjunctive regular path queries. The ApproxRelax prototype, we have described here builds on the theoretical foundations of [16] by implementing for the first time ontology-based relaxation and query approximation for CRP queries. Moreover, ApproxRelax is the first system to provide a visual user interface that allows users to incrementally construct CRP queries and express their approximation and relaxation preferences (this was an area of future work identified in [16]).

6 CONCLUSIONS

Facilitating the collaborative formulation of learning goals and career aspirations has the potential to enhance learners’ engagement with the lifelong learning process. We have described a prototype system called ApproxRelax which provides users with a graphical facility for incrementally constructing search queries over learners’ timelines. We have described how the system can be used to construct conjunctive regular path queries over timeline data and metadata, and to allow approximation or relaxation to be applied to selected parts of the user’s query. We have discussed how such queries are evaluated, showing how the system returns results ranked in order of their distance from the original query, and is able to provide more relevant answers than L4All’s similarity metrics-based approach.

The work that we have presented here is novel both in its aim of supporting lifelong learners in reflecting on their learning and career choices, and also in its technical approach which implements for the first time query approximation and query relaxation techniques for CRP queries, also providing a visual interface for users to incrementally construct their queries and express their approximation and relaxation preferences. Future work includes evaluation of the usability of ApproxRelax with FE/HE learners, and its extension to encompass the missing Qualifications and Industrial classifications, and query templates for Personal and Other episode types. As part of the usability evaluation, we will investigate whether there is a need for the system to provide an explanation of how the overall distance of the query results has been calculated, and how this feedback should be presented to users. Empirical evaluation of our query processing algorithms with realistic volumes of timeline data is still needed, followed by development of appropriate query optimization techniques. A further phase of piloting of the system will then be carried out, prior to exploring the provision of a live service within our institution and more broadly.

REFERENCES

Alexandra Poulovassilis received the MA degree in mathematics from Cambridge University and the MSc and PhD degrees in computer science from Birkbeck. Her research interests center on information management, integration, and personalization. Since 2003, she has been codirector of the London Knowledge Lab, a multidisciplinary research institution which aims to explore the future of knowledge and learning with digital technologies.

Petra Selmer received the BSc degree in computer science from the Rand Afrikaans University (South Africa) and the MSc degree in advanced information systems from Birkbeck. She has been working toward the part-time PhD degree (computer science) since 2008, researching the area of flexible querying of semistructured data. She has held the position of software architect at the Intensive Care National Audit and Research Center since 2004.

Peter T. Wood received the PhD degree in computer science from the University of Toronto in 1989, having previously received BSc and MSc degrees in computer science from the University of Cape Town. His research interests include query languages for various data models, query optimization, active and deductive rule languages, and graph algorithms.