A Tale of Two Entity Linking and Discovery Systems

***Kathryn Mazaitis** <krivard@cs.cmu.edu>,
† Richard C. Wang <rcwang@pagereactor.com>, † Frank Lin <frank@pagereactor.com>,
*Bhavana Dalvi <bbd@cs.cmu.edu>, *Jakob Bauer <jakobbauer@me.com>,
*William W. Cohen <wcohen@cs.cmu.edu>

*School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, USA
† PageReactor.com, Mountain View, CA 94043

Abstract

The long-term research agenda of our group is to evaluate the potential of probabilistic logics for complex, large-scale problems which require data resources naturally encoded as relations. In pursuit of this goal, we compared two systems for performing automated entity discovery and linking in English-language text, as submitted to the 2014 TAC Knowledge Base Population Entity Discovery and Linking (EDL) track. Both systems are based on random-walk strategies for measuring similarity within graphs. The first system is PageReactor, a hand-engineering system originally designed for task of *wikification*. The second is based on ProPPR, a probabilistic logic programming language.

1 Introduction

Modern entity (discovery and) linking systems rely on a variety of approaches to generate and rank candidate KB entities to be associated with a query string, or a "discovered" entity mention, in a text. These approaches typically rely on a number of different large lexical resources, such as: document-length descriptions of KB entities; hyperlinks among entity pages; hyperlinks to entity pages, along with their anchor text; and disambiguation pages. These diverse resources are linked together with a number of natural heuristics, including entity popularity and global coherence of entity assignments, which are often expressed in terms of graph algorithms over the various lexical resources available.

Systems of this sort are challenging to build, test and integrate. One issue is that there is no consensus on the architecture of a entity linking (and/or entity discovery) system. Because the lexical resources above, which are commonly employed in this task, naturally take the form of relations and graphs, rather than (say) numeric features of candidates, it is not obvious how to best decompose the task into a pipeline of well-studied subtasks, such as retrieval and classification.

The long-term research agenda of our group is to evaluate the potential of probabilistic logics for tasks of this sort: complex, large-scale problems which require data resources naturally encoded as relations. The principle technical problem we are seeking to address in this long-term agenda is the tradeoff between efficiency and expressivity for such logics, with the aim of making usefully expressive logics work for practically interesting problems at real-world scales. The KBP entity discovery and linking (EDL) task appeared to us to be a natural testbed to work on. We thus built two systems, one for entity linking (EL) and one for EDL based on ProPPR, a newly-developed probabilistic logic.

We were also fortunate to be able to work with and compare to PageReactor[1], a high-quality hand-engineering system for the closely related task of *wikification*. Wikification differs from EDL in that it does not require classification of entities by type, and does not require "NIL clustering" (i.e., clustering of novel entity mentions by referent); however, being able to observe PageReactor's performance and having access to its designers helped us to pri-

[1]http://enfind.com/pagereactor

Table 1: A simple program in ProPPR. See text for explanation.

```
about(X,Z) :- handLabeled(X,Z)          # base.
about(X,Z) :- sim(X,Y),about(Y,Z)       # prop.
sim(X,Y) :- links(X,Y)                  # sim,link.
sim(X,Y) :-
   hasWord(X,W),hasWord(Y,W),
   linkedBy(X,Y,W)                      # sim,word.
linkedBy(X,Y,W) :- true                 # by(W).
```

oritize the resources we used in our system.

Below, we describe the ProPPR-based system, first presenting results on the 2013 EL task. We then present the overall architecture of the ELD, presenting the changes made to adapt the EL system to the 2014 EDL task, including a description of PageReactor. Finally we present our results and an error analysis, and conclude.

2 The ProPPR System

Some of our submissions were based on ProPPR (Wang et al., 2013), a first-order templating language for defining complex random-walk based algorithms. ProPPR includes learning methods for tuning (although in the EDL submission we used unit weights) and can incorporate arbitrary tabular data easily.

2.1 Background on ProPPR

Below we will give an informal description of ProPPR, based on a small example. More formal descriptions can be found elsewhere (Wang et al., 2013).

ProPPR (for Programming with Personalized PageRank) is a stochastic extension of the logic programming language Prolog. A simple program in ProPPR is shown in Table 1. Roughly speaking, the upper-case tokens are variables, and the ":-" symbol means that the left-hand side (the *head* of a rule) is implied by the conjunction of conditions on the right-hand size (the *body*). In addition to the rules shown, a ProPPR program would include a *database* of *facts*: in this example, facts would take the form *handLabeled(page,label)*, *hasWord(page,word)*, or *linkedBy(page1,page2)*, representing labeled training data, a document-term matrix, and hyperlinks, respectively. The condition "true" in the last rule is "syntactic sugar" for an empty body.

In ProPPR, a user issues a query, such as "?-about(a,X)", and the answer is a set of possible bindings for the free variables in the query (here there is just one such varable, "X"). To answer the query, ProPPR builds a *proof graph*. Each node in the graph is a list of conditions $R_1, \ldots, R_k$ that remain to prove, interpreted as a conjunction. To find the children of a node $R1, \ldots, Rk$, you look for either

1. database facts that match R_1, in which case the appropriate variables are bound, and R_1 is removed from the list, or;

2. a rule $A \leftarrow B_1, \ldots, B_m$ with a head A that matches R_1, in which case again the appropriate variables are bound, and R_1 is replaced with the body of the rule, resulting in the new list $B_1, \ldots, B_m, R_2, \ldots, R_k$.

The procedures for "matching" and "appropriately binding variables" are somewhat cumbersome to describe formally[2], but are intuitive, and illustrated in Figure 1.[3] An empty list of conditions (written $\Box$ in the figure) corresponds to a complete proof of the initial query, and by collecting the required variable bindings, this proof can be used to determine an answer to the initial query.

In Prolog, this proof graph is consructed on-the-fly in a depth-first, left-to-right way, returning the first solution found, and backtracking, if requested, to find additional solutions. In ProPPR, however, we will define a *stochastic process on the graph*, which will generate a score for each node, and hence a score for each answer to the query. The stochastic process used in ProPPR is *personalized PageRank* (Page et al., 1998; Csalogny et al., 2005), also known as random-walk-with-restart. Intuitively, this process upweights solution nodes that are reachable by *many short proofs* (i.e., short paths from the query node.) Formally, personalized PageRank is

[2]Technically, $R_1, \ldots, R_k$ is replaced with $(B_1, \ldots, B_m, R_2, \ldots, R_k) \circ \theta$, where θ is the most general unifier of A and R_1, and $\circ$ indicates applying the substitution θ.

[3]The edge annotations will be discussed later.

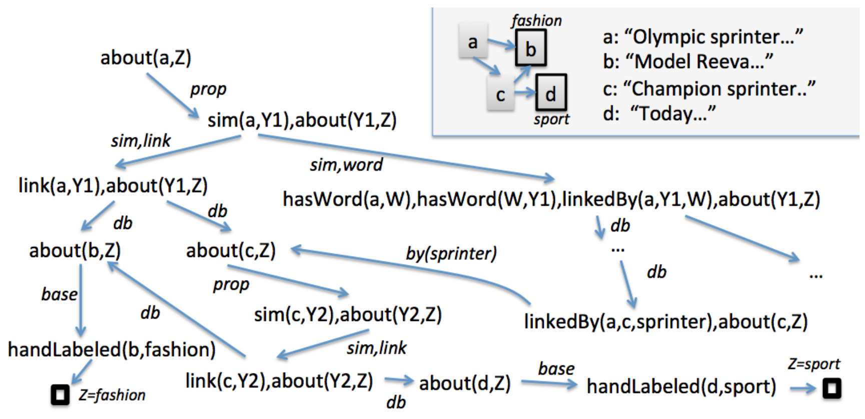

Figure 1: A partial proof graph for the query *about(a,Z)*. The upper right shows the link structure between documents a, b, c, and d, and some of the words in the documents. Restart links are not shown.

the fixed point of the iteration

$$\mathbf{p}^{t+1} = \alpha\chi_{v_0} + (1-\alpha)W\mathbf{p}^t \qquad (1)$$

where $\mathbf{p}[u]$ is the weight assigned to u, v_0 is the seed (i.e., query) node, χ_{v_0} is a vector with $\chi_{v_0}[v_0] = 1$ and $\chi_{v_0}[u] = 0$ for $u \neq v$, and W is a matrix of transition probabilities, i.e., $W[v, u]$ is the probability of transitioning from node u to a child node v. The parameter α is the reset probability, and the transition probabilities we use will be discussed below.

Like Prolog, ProPPR's proof graph is also constructed on-the-fly, but rather than using depth-first search, we use PageRank-Nibble, a technique for incrementally exploring a large graph from a an initial "seed" node (Andersen et al., 2008). PageRank-Nibble takes a parameter ϵ and will return an approximation $\hat{\mathbf{p}}$ to the personalized PageRank vector $\mathbf{p}$, such that each node's approximated probability (under $\hat{\mathbf{p}}$) is within an additive error ϵ of the correct value. PageRank-Nibble also requires time and space $O(\frac{1}{\alpha\epsilon})$, and in particular, will return a proof graph with no more than $O(\frac{1}{\alpha\epsilon})$ edges. This gives ProPPR a fast approximate inference procedure.

We close this background section with some final brief comments about ProPPR.

Scalability. ProPPR is currently limited in that it uses memory to store the fact databases, and the proof graphs constructed from them. ProPPR uses a special-purpose scheme based on sparse matrix representations to store facts which are triples, which allows it to accomodate databases with hundreds of millions of facts in tens of gigabytes. (No more than 35GB RAM was needed for the experiments of this paper.)

With respect to run-time, ProPPR'S scalability is improved by the fast approximate inference scheme used, which is typically an order of magnitude faster than power iteration for moderate-sized problems (Wang et al., 2013), and much faster on larger problems. Experimentation and learning are also sped up because with PageRank-Nibble, each query is answered using a "small"—size $O(\frac{1}{\alpha\epsilon})$—proof graph. Many operations required in learning and experimentation can thus be easily parallized on a multi-core machine, by simply distributing different proof graphs to different threads.

Learning. The personalized PageRank scores are defined by a transition probability matrix W. ProPPR allows "feature generators" to be attached to its rules, as indicated by the code after the hashtags in the example program: for instance, when matching the rule "sim(X,Y) :- links(X,Y)" to a condition such as "sim(a,X)" the two features "sim" and "link" are generated, and when matching the rule "linkedBy(X,Y,W) :- true" to the condition "linkedBy(a,c,sprinter)" the feature "by(sprinter)" is generated. Since edges in the proof graph correspond to rule matches, the edges can also be labeled by features, and a weighted combination of these features can be used to define a total weight for each edge, which finally can be normalized used to define the transition matrix W. Learning can be used to tune these weights to data; ProPPR's learning uses a parallelized SGD method, in which inference on

system	KBP2010 μ-average	B^3 F1	B^3+ F1
ProPPR 2011 test set	0.599	0.950	0.533
Stanford-UBC `nil` (N1), 2011	0.5	0.475	
Stanford-UBC `nil` (N2), 2011	0.5	0.418	
median, 2011		0.716	
best, 2011		0.846	
median, 2013	0.746		0.574
best, 2013	0.833	0.746	

Table 3: Results on 2013 task with 2011 data, relative to published median and best scores. Also included for reference are the Stanford-UBC N1 and N2 baselines from (Chang et al., 2011).

different examples is performed in different threads, and weight updates are synchronized.

Even in the absence of learning, however, it is often the case that ProPPR's personalized PageRank weights will be a useful ranking. (The example program, for instance, propagates labels using the algorithm of Lin and Cohen (Lin and Cohen, 2010) on a hybrid graph containing page and word nodes, and hyperlink and word-to-page edges.) While learning has been used successfully for ProPPR in the past, preliminary experiments suggested that it did not improve performance on these tasks, so it was not used in the final ELD submissions.

2.2 ProPPR on the 2011/2013 Entity-Linking Task

To see if ProPPR was a plausible system to use in KBP 2014, we first developed a ruleset and database using the 2013 task definition and the TAC 2011 KBP English Evaluation Entity Linking Annotation and Queries datasets. The database relations were similar to those described in Section 3.2.1.

- *queryName* is the text of the query.
- *entityName* is the text of the entity name, as encoded in its TAC KB article.
- *anchorText* is the text of inlinks to the entity, as gathered by (Spitkovsky and Chang, 2012), using exact name match to the TAC KB entity name and thresholding the conditional probabilities to $p(\text{entity}|\text{anchor}) > 0.05$ and $p(\text{anchor}|\text{entity}) > 0.001$.
- *inDocument* relates a document to the tokens it contains, and *queryToken* and *entityToken* are analogous for queries and entities.
- *inWikiPage* relates a token to those entities whose disambiguation text contain that token.
- *hyperlink* relates an entity to entities it is hyperlinked to in the TAC KB article for the first entity.

These relations are binary, and are listed in Table 4. (Note, however, that the number of entries per relation in this Table is given for the 2014 corpus. The 2011 corpus is about three times larger, with about 320M *inDocument* triples.

The ruleset is shown in Table 2.[4] The first rules in the program encode specific types of relatedness. For instance, rule 1 states that "If the query text is an exact match for an entity name, return that entity" (which we call the *exact name* strategy). Rule 2 implements the *exact anchor strategy*, i.e., it states, "if the query text is used as linktext to an entity anywhere on the web, return that entity." Rule 3 implements the *hub entity strategy*, i.e., "If text anywhere in the document is used as linktext to an entity, return it and all entities related to that entity." Rule 4 implements a *shared text* strategy, i.e., "If the document shares words with an entity's article page, return that entity", and Rule 5 implements a *soft name match strategy*, i.e., "If the query text shares tokens with an entity name, return that entity."

Rule 5, the soft name match strategy, is an example of the expressive power of ProPPR rules. The personalized PageRank scoring scheme will first spread weight among the tokens of a query such that for longer queries, each token counts for less; then it spreads weight among the entities whose names contain that token, such that frequent tokens (i.e., low IDF tokens like "a, an, the") have less influence in the scoring.

The "related entities" used in Rule 3 are defined by another predicate. In the experiments, we used only Rules 6-7, which consider entities related if their pages are hyperlinked; adding Rule 8 would use another personalized PageRank-style walk to define entity relatedness.

No NIL entities were predicted by the logic program; instead, the top-scoring candidate was selected for each query, without any type checking.

[4]We have added line numbers, for reference below, have simplified slightly by removing one subpredicate that, in the final program used, was defined using only one relation.

Table 2: A ProPPR program for entity linking.

1	answerQuery(Doc,Queryid,EntId) :- queryName(Queryid,Name),entityName(Name,EntId).
2	answerQuery(Doc,Queryid,EntId) :- queryName(Queryid,Name),anchorText(Name,EntId).
3	answerQuery(Doc,Queryid,EntId) :- inDocument(Doc,Name),anchorText(Name,Z),related(Z,EntId).
4	answerQuery(Doc,Queryid,EntId) :- inDocument(Doc,Word),inWikiPage(Word,EntId).
5	answerQuery(Doc,Queryid,EntId) :- queryToken(Queryid,Word),entityToken(Word,EntId).
6	related(X,X) :- true.
7	related(X,Y) :- hyperlink(X,Y).
8	*related(X,Y) :- related(X,Z),hyperlink(Z,Y).*

We assigned a unique NIL id to every query scoring below a threshold (tuned on the training set). We then scored the test set predictions using the 2013 scoring script (`el_scorer.py v0.7`).

This preliminary system produced extremely favorable results relative to the 2011 task, outperforming the best B^3 F1 scores from that year (see Table 3). It performed less well relative to the 2013 task, scoring below the median on both micro-average and B^3+ F1.

This system ran in 25GB RAM and averaged 819ms per query. The fastest query ran in 176ms, and the slowest in 11.431sec.

3 Entity Linking and Discovery

3.1 From EL to EDL

Entity discovery was a new aspect of the task this year, requiring systems to first analyze a complete document and extract regions of the text naming entities to be linked. This required named entity extraction. We used the Stanford CoreNLP NER tagger to do most of the extraction, augmented by a wrapper to exclude quoted text and include relevant metadata (specifically, author annotations). We also refined our linking approach, based on error analysis on the 2014 development set. Notable additions included query aliases, tokenized query lookups into the anchor text table, and local context at the sentence level.

For the purpose of comparison to other systems, we included an alternate procedure to generate proposed entity links, based on PageReactor. PageReactor was augmented by a postpass to assign named-entity categories to linked pages (which was not part of the original system.)

Finally, since our EL did no NIL clustering, we added a NIL clustering module, and explored the use of some relatively sophisticated clustering methods for this task, based on research done elsewhere at CMU. The same NIL clustering methods were tested with PageReactor and the ProPPR-based system. The overall architecture of our system is shown in Figure 2.

3.2 ProPPR for the 2014 Entity Discovery and Linking Task

3.2.1 Database

relation	arg1	arg2	size (E87)
anchorText*	string	eid	1,214,052
entityName	string	eid	818,698
entityToken	token	eid	2,116,658
entityType	eid	type	818,741
extractedType	qid	type	3,953
hyperlink	eid	eid	2,397,645
inDocument	did	token	26,098
inSentence	sid	token	43,385
inWikiPage	token	eid	105,942,944
queryAlias	qid	string	3,287
queryName	qid	string	3,951
querySentence	qid	sid	3,953
queryToken	qid	token	14,753

Table 4: Relations in the linking database. *anchorText* was excluded from the limited submission.

The database used for ELD included the following relations, in addition to the ones used for the linking task. The *entityType* relation gives the type of an entity, as encoded in its TAC KB article. The *extractedType* relation gives the type of a query string, as extracted by the Stanford NER tagger. The relation *querySentence* gives the ID of the sentence containing a query, and the relation *inSentence* gives

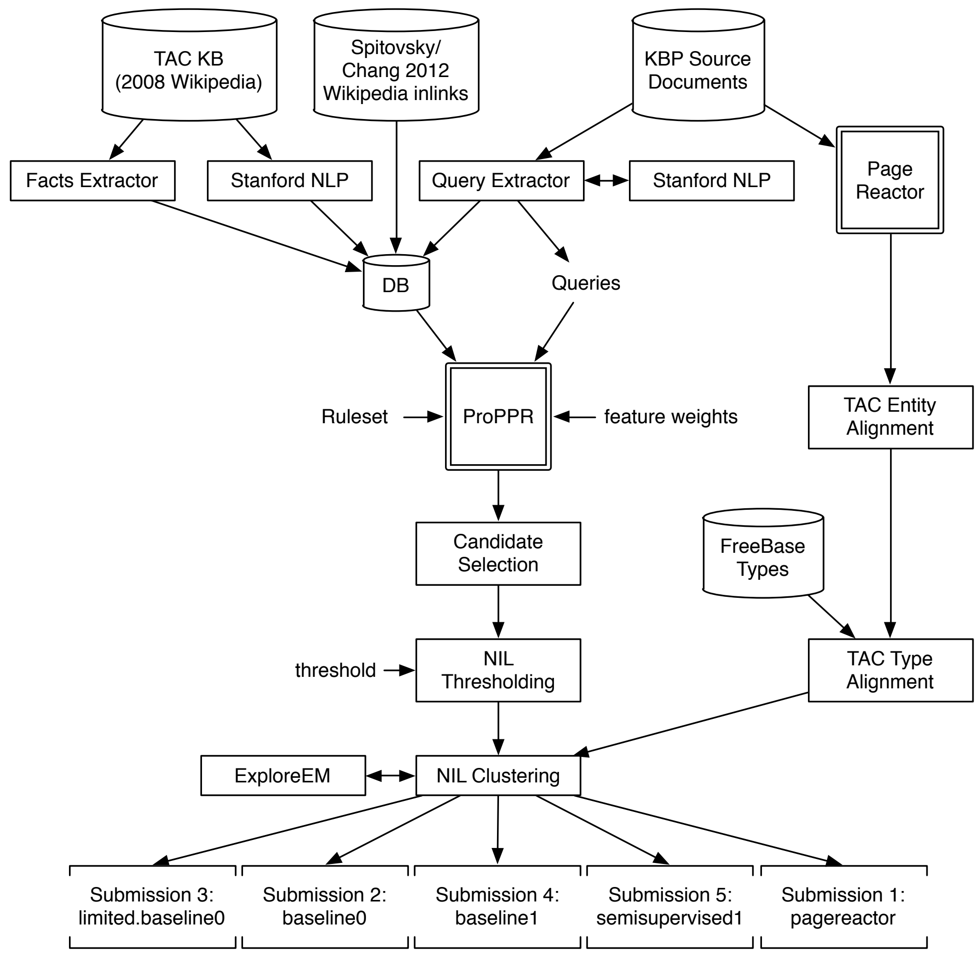

Figure 2: Combined system diagram for the CMU submissions

the tokens in a sentence given its ID.

Finally, the *queryAlias* relation links a query string q to the text of the first query in the document to include this query's text as a substring. This was used for a simplified coreference resolution to handle cases where, e.g., "Casey Jones" is used as the first entity referent in a document, and only "Jones" thereafter.

As noted the *anchorText* relation was not used in the limited submission.

3.2.2 Logic Program

The logic program executed by ProPPR for the EDL task took the form of a list of strategies similar to those used for EL, but with the addition of conditions specifying *typeConstraints*—learned conditions on matches between the entity type and the NER-extracted type. For example, extract match strategy was extended to:

```
answerQuery(Doc,Queryid,Entityid) :-
    queryString(Queryid,Name),
    entityName(Name,Entityid),
    typeConstraint(Queryid,Entityid).
```

The program was structured so that anywhere the query text was used, the query alias could be substituted instead. This allowed for better coreference resolution within a document. Two more strategies were added, one using sentence-level rather than document-level context, and one that allows the NIL entity to be proposed regardless of type constraints. There were eight strategies (i.e., rules for the *answerQuery* predicate) and 18 total rules in the ProPPR program.

While ProPPR permits the use of learning methods to tune the weighting of different strategies in the logic program, for this task we did not see improved performance using trained weights. Engineering features for the entity linking problem that respond well to training is a priority for future work.

3.2.3 Candidate selection

ProPPR uses the logic program to generate a list of candidate links for each of the queries identified in the discovery phase—-as many as 1000 such candidates for this particular application. To select the best candidate link for each query, we take the highest-scoring non-NIL link whose entity type matches the extracted type for the query, or if all criteria cannot be satisfied, the highest-scoring non-NIL link, or else the highest-scoring link. We then assign NIL to all responses whose score falls below a threshold. This threshold was tuned on the development data set.

3.3 PageReactor

PageReactor uses string-processing methods (tries) to recognize candidate entity mentions, and builds a graph connecting them to possible referents. Edges in the graph are weighted based on corpus statistics from a 2014 Wikipedia, and a random-walk algorithm (Wang and Cohen, 2007) is used to rank the referents. PageReactor, a prototype commercial system, is carefully tuned for this task, but includes no NIL-clustering tools: we created a new NIL cluster for each 2014 entity that could not be aligned to a TAC (2008) entity. A second postpass is used to assign TAC types to entities, using heuristics based on FreeBase types.

In more detail, PageReactor[5] is a real-time system for linking of entities to knowledge-bases in text documents. PageReactor is composed of four main components: 1) the learning module, 2) the named-entity recognition module, 3) the entity disambiguation module, and 4) the service module.

The learning module is responsible for running batch data processing and statistics-gathering jobs over the training data, which typically consists of large corpora of documents and external knowledge-base dumps. For this entity-linking task, we used the May 2014 dump of Wikipedia (wikipedia.org) pages to train PageReactor's learning module.

The learning module learns the entity names, their synonyms, and how likely they are being linked to its Wikipedia page when mentioned. For example, we would link the abbreviation "THEM" to the TV series "The Hidden Extreme Magic" only when the surrounding text mentions things like TV shows and magic tricks. The output of the learning module includes statistics about the frequency of the occurrence and co-occurrence of entities in the training documents and the relationship between different entities.

The named-entity recognition module, when given a query document (in the form of a string of plain text characters), scans through the document, and annotates every entity name that it recognizes by utilizing the statistical model learned from the training corpora. Each annotation is then mapped to all possible knowledge-base entities it may refer to (e.g., Lincoln is mapped to both the United States president Abraham Lincoln, the town car Lincoln manufactured by the Ford Motor Company, and several middle schools and high schools in the US.).

From the recognized entities, the entity-disambiguation module instantiates a small graph for this document. The difference between this document-based graph and external knowledge-bases such as Wikipedia and Freebase is that relationships between entities in this graph are not ontological (e.g., for answering questions like "is *Mark Melancon* a team member of the baseball team *Pittsburgh Pirates*"?) but statistical, based on the occurrence counts in the training documents. For instance, an edge might connects the entities above if, when the words "Melancon" and "Pirates" appear in close proximity to each other in a document, it is likely that "Melancon" refers to the baseball player *Mark Mecanon* and "Pirates" refers to the baseball team *Pittsburgh Pirates*. The final assignment of tokens to entities is jointly inferenced using a random graph-walk algorithm on this internal graph, similar to that used in prior work by Wang and Cohen (Wang and Cohen, 2007). Figure 3 illustrates two constructed graphs, one for the short document "Jaguar is a luxury car" and "Jaguar is an aggressive feline", respectively. The text token "Jaguar" can possibly link to many

[5]pagereactor.com

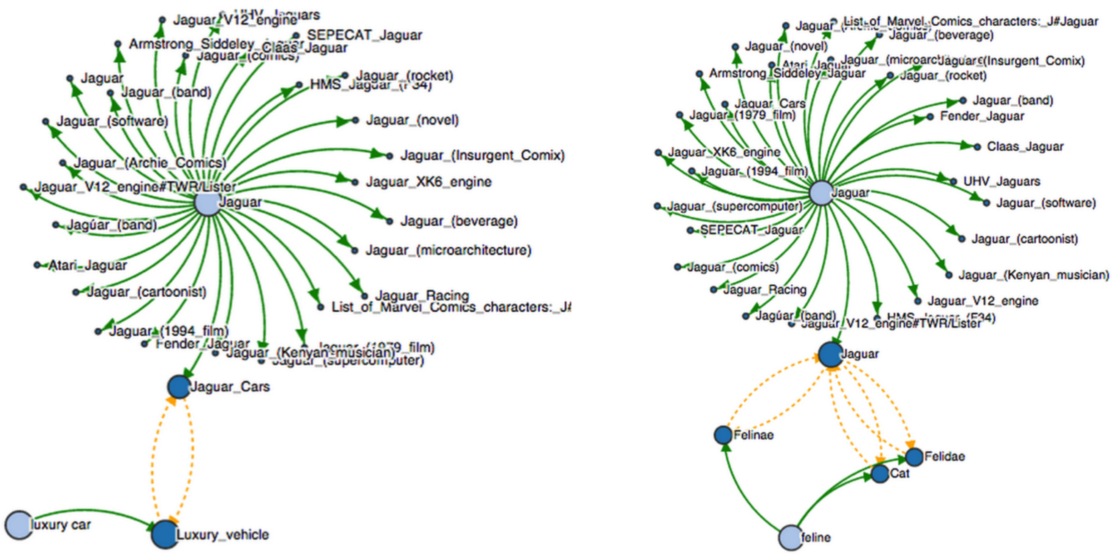

Figure 3: Disambiguation graphs constructed by PageReactor from two short documents: "Jaguar is a luxury car" (left) and "Jaguar is an aggressive feline" (right).

Wikipedia entries (light blue circles are text tokens and point to entity candidates), but the presence and absence of the text tokens "luxury car" and "feline" helps the recognition module to determine the best candidate.

As a post-processing step for the entity-linking task, we collected a mapping of Wikipedia URL to entity types (eg. albums, person, film, location, books, etc.) from the August 2014 Freebase dump. This information is used in the post-processing step to filter out entities that have types irrelevant to this KBP task.

The service module wraps all above-mentioned modules so queries can be made via a RESTful HTTP endpoint. Besides optimizing the parsing of incoming HTTP queries and delivery of output JSON results, it is also responsible for preprocessing of input text and determining certain inference parameters to the modules based on meta-information of the input text (e.g., document language, document type, document length). PageReactor is also available to the public via the service module.[6]

[6]More information and online demos are available at www.pagereactor.com; those interested in using the PageReactor API can request an API key via info@pagereactor.com.

3.4 NIL Clustering

We implemented several different NIL clustering methods. Baseline 0 and Baseline 1 group together identical strings, and identical strings in the same document, respectively. Both baselines use the exact text for the query, ignoring the query alias. We also used a more sophisticated k-means based SSL method which introduces new clusters as needed, guided by AICC measures of complexity (Dalvi et al., 2013). We included the query tokens, tokens from the query sentence, and the full list of candidate scores as features. This method was seeded with NIL clusters identified by the PageReactor system for queries matching those identified by the ProPPR system's discovery phase.

Unfortunately, the more complex NIL clustering did seem to afford any improvements over the simpler baselines in the development data, so based on results, we used Baseline 0 for the limited submission.

4 Results on EDL

Although the PageReactor system scored above the median for precision on the evaluation EDL data set, the results overall were disappointing. As an error analysis, we examined 100 random query responses from each system for manual error analysis. The

	development (E54) v1.1 2014				evaluation (E87) 2014			
system	WikiF1	CEAFmP	CEAFmR	CEAFmF1	WikiF1	CEAFmP	CEAFmR	CEAFmF1
best					0.678	0.750	0.712	0.730
median					0.509	0.480	0.670	0.559
PageReactor	0.338	0.277	0.211	0.240	0.422	0.611	0.272	0.377
ProPPR:limited.baseline0	0.178	0.689	0.238	0.354	0.144	0.308	0.232	0.265
matching subset	0.378	0.587	0.584	0.585				
ProPPR:baseline0	0.189	0.693	0.239	0.356	0.143	0.307	0.232	0.264
matching subset	0.396	0.585	0.581	0.583				
ProPPR:baseline1	0.189	0.681	0.235	0.350	0.143	0.301	0.227	0.259
matching subset	0.396	0.584	0.580	0.582				
ProPPR:semisupervised1	0.189	0.337	0.117	0.173	0.143	0.193	0.146	0.166
matching subset	0.396	0.314	0.312	0.313				

Table 5: CohenCMU results on the EDL task. Due to labeling issues with development dataset v1.1, we computed scores relative to the entire set of gold labels, and to the subset of gold mentions located by ProPPR's discovery phase.

	diagnostic (R51) 2014		
system	all	in KB	NIL
best	0.821	0.796	0.855
median	0.698	0.648	0.767
ProPPR:baseline0	0.276	0.159	0.412
ProPPR:baseline1	0.273	0.159	0.406
ProPPR:semisupervised1	0.152	0.159	0.083

Table 6: CohenCMU B^3+ F1 results on diagnostic data set (perfect discovery)

count	**type**
17	substring
11	PER
4	GPE
2	ORG
7	mistaken identity
6	team or city
3	non-sequitor
33	total

Table 7: Linking errors in 100 random ProPPR responses. See text for explanation.

count	**type**
2	NIL:PER $\rightarrow$ NIL:ORG
1	NIL:GPE $\rightarrow$ NIL:PER
1	NIL:GPE $\rightarrow$ NIL:ORG
1	substring (ORG)
5	total

Table 8: Linking errors in 100 random PageReactor responses. See text for explanation.

results are tabulated in Tables 7 and 8. Our hand-labeled sample overestimates the precision of our systems, but was done identically between the two and is thus a valid metric for comparing them.

For this analysis, at 95% precision compared to ProPPR's 67%, PageReactor leads the two systems in accuracy. The majority of the PageReactor errors are regarding the type of an entity not found in the TAC KB. For example, the query "Boe" was assigned a NIL entity of type GPE by PageReactor, when in the source document "Boe" is short for a person named "Roy Boe" and thus should have received type PER.

The most common linking error in the ProPPR system is one where the query and entity name share a common substring, but this substring is not enough to fully identify the entity. For example, ProPPR answered a query for "Edgar Velasquez" with an entity named "Nick Velasquez." This issue arises because, while the ProPPR system does personalized PageRank scoring does understand the value of high-IDF tokens, it has no understanding of name morphology; in particular, it treats last and first names identically, rather than learning the relative weight for matching a given name compared to a surname. Permitting substring matches is important in many cases, such as when the entity name contains a little-used middle name, but we need to leverage additional information in order to determine when the presence of non-matching text should penalize a candidate link. In the PageReactor system, we saw a similar error occur when the query "Navy" was linked to the entity for the Naval Academy instead of the US Navy.

The second most common linking error in ProPPR concerns mistaken identity; when the entity name and the query text match, but a different entity is meant. The ProPPR system notably predicted Hoover (the seal) for a query that should have referenced Hoover (the president), and cricket player Bill O'Reilly for a query that should have referenced the political commentator. We predict that many of these errors could be addressed by considering a prior over entity pages as a function of their number of inlinks, since the president and the political commentator are likely to be mentioned more often than the seal or the cricket player—an additional step of feature engineering that would be straightforward to perform. In principle, improved weights for the consideration for the similarity of the vocabulary of the entity's disambiguating text with the text of the document should also help with this problem.

There are also a number of ProPPR errors involving the notorious problem of determining when a city name is being used as a reference to a location and when it is being used to stand in for a sports team.

The 2014 ProPPR system required more memory than the preliminary 2013/2011 system due to the increased size of the database for local context information, but it still ran in a respectable 35GB RAM. Each query took on average 337ms to run, with the fastest query running in 97ms and the slowest in 9.325sec.

4.1 ProPPR Strategy Analysis

As an additional post-task analysis, we evaluated the performance of the ProPPR system using modified rulesets that excluded each non-nil strategy one at a time. For simplicity, instead of using an exact threshold, we assigned NIL to the bottom 60% of responses for each variant, and used a unique nil id per query. The linking performance (`strong_typed_link_match`) of each variant is displayed in Table 9. We expect the most important strategies to have the greatest negative effect on the linking score when they are left out. Thus, in descending order of importance, we have the strategies exact name, anchor query string, document-level shared text, hub entity, sentence-level shared text, anchor query token, and token in name. For this dataset, eliminating the token in name and anchor query token strategies would have resulted in a higher score than we achieved on the evaluation.

excluded strategy	WikiP	WikiR	WikiF1
-	0.210	0.115	0.149
exact name	0.161	0.089	0.114
token in name	0.228	0.121	0.158
anchor query string	0.197	0.109	0.140
anchor query token	0.212	0.117	0.151
hub entity	0.207	0.115	0.148
shared text (document)	0.205	0.108	0.141
shared text (sentence)	0.210	0.115	0.149

Table 9: Wikification/linking performance on ruleset variants excluding each one of the seven non-nil strategies.

5 Conclusion

Multistage NLP systems are needed to perform complex tasks such as ELD, and systems of this sort are challenging to build, test and integrate, particularly when they are based on many statistical resources which naturally take the form of relations and graphs, rather than numeric features. In this paper we sought to evaluate the potential of a new, efficient probabilistic logic called ProPPR for tasks of this sort. In particular we built and evaluated two systems, one for entity linking (EL) and one for entity linking and discovery (ELD). We also compared these to PageReactor, a high-quality hand-engineering system for the closely related task of *wikification* which used similar resources.

Our experience can be summarized as follows. The logic-based systems clearly do scale to problems of this size, running queries in a third of a second, on average, on a high-end desktop machine, even over databases with hundreds of millions of tuples. The implementation is concise, and it is relatively easy to explore variants of the basic algorithm. Further, the logic-based systems performed well on the simpler task of EL.

Unfortunately, it is quite difficult to perform detailed error analysis of results provided by ProPPR—perhaps because they combine information from many sources to derive an overall score. As a consequence it was very difficult to debug logic programs evaluated on large-scale problems, a disadvantage which ultimately negated any engineering advantages potentially offered by the architecture. This problem was compounded by the failure of learning to improve performance on either the EL or ELD task, an issue which is being investigated at the time of this writing, but far from being resolved. Hence one important conclusion is that scalability is not enough for practicality—instead, it seems likely that better tools for monitoring and visualizing performance are needed to make tools like ProPPR practically useful for large-scale tasks.

Acknowledgements

This work was sponsored in part by DARPA grant FA87501220342 to CMU under the DEFT program, and a Google Research Award.

References

Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. 2008. Local partitioning for directed graphs using pagerank. *Internet Mathematics*, 5(1):3–22.

Angel X. Chang, Valentin I. Spitkovsky, Eneko Agirre, and Christopher D. Manning. 2011. Stanford-UBC entity linking at TAC-KBP, again. In *Proceedings of the Fourth Text Analysis Conference (TAC 2011)*, Gaithersburg, Maryland, USA, November.

Kroly Csalogny, Dniel Fogaras, Balzs Rcz, and Tams Sarls. 2005. Towards scaling fully personalized PageRank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2(3):333–358.

Bhavana Dalvi, William W. Cohen, and Jamie Callan. 2013. Exploratory learning. In *ECML/PKDD*.

Frank Lin and William W. Cohen. 2010. Semi-supervised classification of network data using very few labels. In Nasrullah Memon and Reda Alhajj, editors, *ASONAM*, pages 192–199. IEEE Computer Society.

Larry Page, Sergey Brin, R. Motwani, and T. Winograd. 1998. The PageRank citation ranking: Bringing order to the web. In *Technical Report, Computer Science department, Stanford University*.

Valentin I. Spitkovsky and Angel X. Chang. 2012. A cross-lingual dictionary for English Wikipedia concepts. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012)*, Istanbul, Turkey, May.

Richard Wang and William W. Cohen. 2007. Language-independent set expansion of named entities using the web. In *ICDM-2007*.

William Yang Wang, Kathryn M. Mazaitis, and William W. Cohen. 2013. Programming with personalized page-rank: A locally groundable first-order probabilistic logic. In *CIKM-2013*.