BioGraph

UC Berkeley School of Information

Masters Final Project Report

May 2011

Krishna Janakiraman and Sean Marimpietri

Contents

1 Introduction **2**
- 1.1 Problem . . . 2
- 1.2 Approach . . . 3

2 Resources **5**
- 2.1 SNAC . . . 5
- 2.2 Freebase . . . 6
- 2.3 IMDB . . . 7
- 2.4 Discussion of the Resources . . . 7

3 Schema Matching Tool **9**

4 Record Linkage **11**
- 4.1 Method . . . 12
  - 4.1.1 Preprocessing . . . 13
  - 4.1.2 Indexing . . . 13
  - 4.1.3 Finding Candidate Entity Matches . . . 14
  - 4.1.4 Supervised Learning for Predicting Entity Matches . . . 14
  - 4.1.5 Postprocessing . . . 18

5 Visualization **20**
- 5.1 Related Work . . . 22
- 5.2 Graph as query language . . . 23
- 5.3 Interface Components and Methods . . . 24
  - 5.3.1 Query Graph Interface . . . 24
  - 5.3.2 Searching for Subgraphs . . . 24
  - 5.3.3 Result Graph Visualization and Interface . . . 27
- 5.4 Evaluation . . . 30

6 Conclusion **32**

Abstract

Cultural institutions such as museums and libraries house large collections of biographical information in the form of archival context and name authority file records. However, today these institutional collections are complemented by sizable, but less authoritative, crowd-sourced online resources such as Freebase and IMDB. These collections contain significant information that is either not easily accessible through or not present at all in the institutional collections. BioGraph is the result of our effort to connect and enrich institutional collections with such online resources and to provide tools to explore the latent social network present in these biographical collections.

In BioGraph, we have designed and implemented binary classifiers to merge and de-duplicate entities across diverse collections. As well, we have designed and implemented a visual query interface that allows users to intuitively express structural queries and explore social networks of the entities in our combined biographical collection.

More specifically, we use these techniques to enrich the biographical information generated by the Social Networks and Archival Context Project (SNAC) which includes curated information from the Library of Congress, the Online Archive of California and other cultural institutions, by connecting it with the Freebase and IMDB collections. We report on the challenges faced in merging these diverse collections and discuss how our visual interface can be used to query the rich social network information present in the combined and enriched collection.

Acknowledgments

We thank Prof. Ray Larson, and the SNAC project, for giving us access to the SNAC dataset and for providing us with compute and storage infrastructure.

Chapter 1

Introduction

1.1 Problem

With the emergence and spread of the Internet in the past decades, scholars and laypersons alike have been presented with new alternatives that bypass the traditional nodes through which research takes place: libraries. Obviously, conducting research by making use of libraries has never guaranteed a yield of high quality information. Quality of research inevitably varies from researcher to researcher. However, one great virtue of libraries as access points for research is their attention to issues of authority and control. Contrastingly, this is generally not the great virtue of web based resources, where crowdsourcing, scale and specialization of content drive the value of a resource.

Independent of web resources, efforts are underway to aggregate library resources and make them more widely available. The Social Networks and Archival Context (SNAC) project is addressing the "ongoing challenge of transforming description of and improving access to primary humanities resources through the use of advanced technologies."[1] The challenge here is that "because archivists have traditionally described records and their creators together, this information is tied to specific resources and institutions." As such, a central goal of the SNAC project is to "use digital technology to "unlock" descriptions of people from finding aids and link them together in exciting new ways."

The result of this is an aggregated collection made available through a prototype "historical resource and access system." A great wealth of information about historic public figures, derived from finding aids to primary humanities resources, can be found in this collection. Understandably, the information in the SNAC collection is rich in those facets, such as alternate names, existence dates and links to related *works* associated with the entities, that are of foremost interest to librarians and archivists, whose use of the information is to locate the primary humanities resources. However, the collection has significant gaps in other important facets of biographical information, such as gender, occupations (Figure 1)

[1] http://socialarchive.iath.virginia.edu/index.html

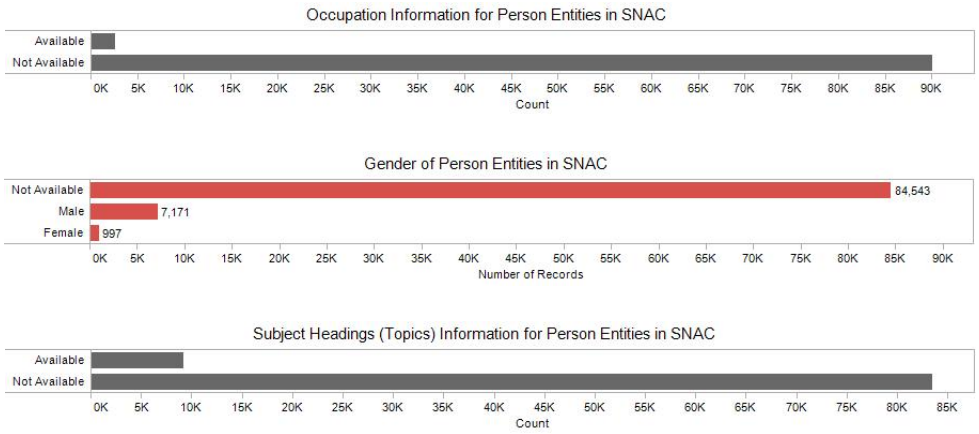

Figure 1 Extent of missing data in SNAC

The SNAC collection is uniquely rich in information on the social network, derived under archival context, among the historic public figures mentioned in these humanities resources. This is because archivists, while describing the entities mentioned in an archival record, have taken great effort in *linking* the entities mentioned in the records. The SNAC project has gone a step further by aggregating and merging the entities and their corresponding social networks across collections of such archival descriptions. The result of which is a biographical collection of historic public figures rich in information on social networks derived from archival contexts.

In order for the unique social graph information, housed by the SNAC collection, to have a wider reach and access it is necessary to fill the core information gaps in the collection. Furthermore, the current exploratory search interface, though ideal for finding information about single entities or entities grouped under facets, are not ideal for exploring and discovering latent social structures present in the SNAC collection.

1.2 Approach

Our approach to address the information gap problem is to develop algorithms that automatically incorporate information available in the Freebase[2] and IMDB[3] web resources into the SNAC collection, thereby enriching it. We provide our rationale for choosing Freebase and IMDB in the next chapter and in detail discuss these three collections. In short, these three collections rightly complement the SNAC collection and are varied across many dimensions such size, domain and ontology design making our approach generalizable towards enriching similar other sets of biographical collections.

Towards providing a better interface to explore the social graph of the enriched collection, we have developed a query by example interface that allows users to directly express social structures they are interested in discovering as a *query graph*. We first provide brief introductions to the three main parts of our approach: matching schemas between resources,

[2]http://www.freebase.com/
[3]http://www.imdb.com/

merging entities between resources, and development of a graph-based, visual query interface for accessing the merged collection.

The motivation for developing a schema matching tool emerged from our concern about the costs, in terms of time and sanity, which are associated with manual review of large sets of metadata in collections. Of specific concern to us was how we would effectively review Freebase to determine those properties that might make a significant contribution to the aggregated collection. With this concern and task in mind, we implemented a simple schema matching tool that, for a given field in one resource and given suggested search terms, would return fields in Freebase that might be a match for that field. The tool uses the hierarchical structure, metadata terms and field instances to compute suggested relevant matches.

Matching the schema of the SNAC collection with those of Freebase and IMDB provides a way to compare the entities in these three collections. However, a major challenge in linking multiple biographical collections is to automatically find the entities that could be linked across collections. Unlike the schema matching step where we approached the problem in a semi-automatic way, the sheer number of entities in each of the collections prevents any manual, curation-based effort to link entities. This motivated us to develop supervised machine learning techniques that automatically find entities in Freebase and IMDB that could be matched with a given SNAC entity.

The last and a very important part of our approach is the development of the visual query interface for mining structures in the underlying social graph. SNAC has deployed a prototype interface for exploring its collection, which allows users to navigate entries by way of a search bar and hyperlinks. However, we were interested in exploring interfaces that accept more complex queries through which users can directly express the social structure pattern they are interested in discovering. At first, we considered the possibility of using a question answering(QA) system, but this is nontrivial to implement and certain types of questions are not best represented in a textual way. There are specialized query languages like SPARQL and Freebase's MQL but these query languages requires the user be able to formulate complex queries with the language, which is difficult for the user. Therefore, we ultimately settled on developing a visual query interface where complex structural queries can be expressed in an intuitive way by maneuvering nodes, edges and attributes in formulating a query graph. We have developed a subgraph search algorithm and interactive graph visualization techniques to discover and visualize the subgraphs in the underlying social graph database that exactly match the query graph provided by the user.

Chapter 2

Resources

Before discussing our approach to the problem described above, here we introduce SNAC and the resources we connect and use to enrich SNAC. These resources are Freebase and the Internet Movie Database (IMDB). Taken together with SNAC, the organizing principles of these three collections exist along a continuum, from strong authority and control on one end to a very open crowdsourcing model on the other, and fall on three different points on this continuum. Additionally, these collections can be thought of as existing along a second continuum at three different points from highly person-oriented collections on one end (SNAC) to highly thing-oriented collections on the other end (Freebase). Developing an understanding of how the three collections vary with regard to curation practices, purpose, content, structure, and presence of biographical information is an important part of developing the schema matching tool introduced above. Familiarization with the collections helps motivate approaches to developing the schema matching algorithm. And further, the diverse collection types reviewed can be taken as a sample of possible types of collections, which is useful for developing the schema matching tool for the general, generic case.

2.1 SNAC

As mentioned above, SNAC, like the merged collection we create, is a merged collection itself, which is derived from various archival resources, including the Library of Congress, the Online Archive of California and others. SNAC distinguishes between three high level types, corporate bodies, persons, and families, with an emphasis on biographical information and relationships between entities derived from descriptions of records of specific resources. It is straightforward to see that "person" is the least common denominator here among the high level types, which motivated us to conduct our analysis at this level of persons and biographical information. We do not directly consider the corporate body and family types here.

There are various types of information present in SNAC, some of which are given a structured representation with XML and some of which occur as more or less free text. There are core fields for a given person entity that correspond to the following categories: name, birth

date, death date, gender, occupation, and nationality. The actual field names are not as explicit, given they are designed to incorporate family and corporate body entities. For example, birth date is encoded as "fromDate" and death date as "toDate". As well, there is a subject field, which is derived from the Library of Congress subject headings. Then, SNAC contains information about other entities with whom/which an entity has "corresponded with", is "associated with", and documents which the entity is the "creator of" or is "referenced in". Finally, there is a section called "Biographical history" that may be associated with an entry, which is generally ordered by date, lacks metadata and contains free text. This is the range of content, fields, and relationships that can be expressed about any person entity in or any relationship between a person entity and another entity in SNAC. As we can see, SNAC does not employ a wide range of facets to encode biographical information. This is due to the fact that the EAD/EAC-CPF[1] standards, which SNAC uses to derive its data, were designed with the goal of finding and locating resources and not representing their content.

2.2 Freebase

The next resource we utilize in our project is Freebase. Like SNAC, Freebase is an aggregated resource. However, unlike SNAC, which has been curated and managed by professional archivists and librarians, Freebase is heavily crowdsourced and allows its users to contribute to its maintenance. In fact, Freebase not only allows users to contribute content to the collection but also allows them to shape the structure of the collection by creating new schemas with their own types and properties.

Web resources from which Freebase data is derived include Wikipedia, MusicBrainz and WordNet, but there are many, many others. As such, Freebase stands in strong contrast to SNAC, as the SNAC collection preserves the standards of authority and control present in the institutions from which its resources are derived while Freebase opens not only its content but its structure as well to users for contribution.

Given Freebase derives its content from so many resources and allows users to contribute to both its structure and its content, it is not surprising that Freebase covers a vast range of topics. In fact, the range of content in Freebase is so broad it is not even really possible to describe in the same way we have described the SNAC collection above. At a high level, Freebase covers topics as diverse as cricket, music, physics, time, symbols and amusement parks. At a granular level, it contains fields as specific as /meteorology/beaufort_wind_force/wave_height, /chemistry/isotope/half_life, ... The range of content in Freebase is one of the defining characteristics of the collection.

When we turn our attention to the biographical information present in Freebase we find a variety of properties within two types, people/person and people/deceased_person: date of birth, place of birth, country of nationality, gender, profession, religion, ethnicity, parents,

[1] http://www.loc.gov/ead/

children, siblings, spouse (or domestic partner), employment history, education, users who say they are this person, signature, height, weight, quotations, places lived, age, notable professions, and languages. There are other person types in Freebase in addition to /people/person and /people/deceased_person, but these two types appear to be the only types that contain properties that prove useful for the purpose of enriching the SNAC colletion.

2.3 IMDB

The last collection we consider is IMDB (the Internet Movie Database). As with SNAC and Freebase, we chose IMDB because it is an excellent source of biographical information. However, it is also interesting for our analysis because it occupies another point on our continuum between SNAC and Freebase. Historically, IMDB was generated manually by film enthusiasts, but as with Freebase, IMDB users may now contribute to the resource by adding content. However, when IMDB users contribute to the resource all of their contributions must first be filtered by IMDB officials who review the contributions before approving them in effort to maintain a standard of quality for the resource. With regard to the authority and control, crowdsourcing continuum, this puts IMDB in between SNAC and Freebase.

IMDB holds a great deal of content that ranges across actors, actresses, directors, film, TV, news and people, including specifics such as plot keywords, release dates, budgets, cinematographic process, etc. In this way, IMDB has thorough coverage of the film-based entertainment industry space. Focusing in on the domain of biographical information within IMDB, we found the following fields: name, real or birth name, nickname, biography text, date of birth / birth location, height, trivia, date of death / death location / cause, spouse / marriage date / location / # children, trademark, where person is now, other works, biographical publications, quotes, salary.

2.4 Discussion of the Resources

The three resources we have selected for our project derive their content from various types and numbers of resources. They employ practices that range from strict standards for authority and control to very open use of crowd sourcing approaches. Surely there are many other types of collections we could have used in our project, but we believe these three provide sufficient diversity for our purposes.

To develop a frame of reference against which we can compare the three resources, it is useful to consider intuitively what properties of any person are logical (necessary) and simple (probable) where a logical property is a property that necessarily must be associated with any person, and a simple property is a property that is probable to be associated with many persons but one that is not necessary. Any such ontological claims about what necessarily exists are dependent on certain concessions, and here we limit "persons" to the common notion of subjects of today's nation-states, born into public territories and given legal names,

and force out of mind more anthropological and philosophical complexities. Given this, we have the following (imperfect) logical properties for any person: [great][grand]father, [great][grand]mother, birth place, death place, birth date, death date, name, sex/gender; and the following (imperfect) simple properties: sibling, [great][grand]child, parent of child, spouse/partner, occupation, nationality. These properties are imperfect, or problematic, not only because they are anthropologically and philosophically naive, but also because they do not address issues of attribution and representation.

Considering the two continuums and how directly the fields/properties in the three resources map to our set of logical/simple properties of "person", based on this limited evaluation, SNAC's fields bore a high similarity to our set, despite also being fit for family and corporate body types. Even though the field names were different at times, the type of fields closely matches the properties mentioned above. As well, Freebase had fields that fit our properties of "person" quite closely, though it did have several additional properties. However, IMDB, existing in the middle of the continuums, had logically awkward fields.

This could be for a variety of reasons, but we hypothesize that properties of IMDB's category "person" were so messy because that collection is centered on film, a specific type of thing, separate from the concept of "person" and the abstract concept of "thing." Compare this with how SNAC is centered on persons, families and corporate bodies, the last two of which are reducible to the fundamental type "person." IMDB is a resource that is largely centered on persons, but here again, as with the first continuum, IMDB sits in the middle between SNAC and IMDB. And finally, consider how Freebase is oriented toward the abstract notion of a "thing." IMDB being in the semantic middle ground with regard to the concept of "person" may force a messy chop of the categories of "person" for this resource.

Chapter 3

Schema Matching Tool

Now that we have discussed the motivation for our project, explained the resources we use, and outlined our approach to the project, we can turn our attention in more detail to the first of the three major components of our approach: the schema matching tool.

Depending on the type of collection and the principles that govern its organization, the metadata that describe its content may differ widely. However, when metadata is used to enclose similar concepts, it makes sense that the terms of the metadata in each schema should bear some resemblance to one another.

It would be foolish to imagine an algorithm that is capable of matching schema fields from different collections in a completely automatic fashion. Agreement over naming rules and ontology definitions has been a great challenge to achieve even among professionals, and it is hard to imagine there will ever be consensus here either. Given these inevitable limitations, we aimed to design a tool that does not attempt to make positive matches between schema fields, but instead suggest fields that are most likely to be a match.

There are a few aspects of the schema and data that can be used to develop such an algorithm[11]. First, there are the field names that occur in the hierarchy: for example, "/people/person/nationality" in Freebase. Each of these terms: *people, person, and nationality*, might be of use for the algorithm. Second, there is the hierarchy itself; that is, which of these terms are parents or children of the others, what types of terms occur at what levels of the hierarchy, etc. In practice, it is difficult to make use of information about the hierarchy of metadata terms, and we have excluded it from the design of our tool. And lastly, there is the instance of the field itself. If the instance contains only letters of the Roman alphabet, only numbers, or some other pattern that may be of some significance, this information can help predict the type of field. In our approach we only used information on the field names.

Next, there is the question of how the fields from each schema can be compared. The central idea is to represent a field as a vector in a high-dimensional vector space. Two fields which are closer will be closer to each other in this vector space. The vector space model of information retrieval considers each document as a vector that is defined by the terms in the document. If we take a field from the schema, there will be terms at various levels in the hierarchy of the metadata and in the instance. These terms are grouped together and used

to define the field vector. In essence, instead of tf/idf for terms in documents, we have tf/idf for terms in fields.

Once the information from the field terms are used to represent the fields as vectors, the fields can be compared by comparing their respective vectors using any distance metric. We experimented with both Euclidean distance and Kullback-Liebler divergence. In the latter case, each vectors represented the distribution of terms in the field names. In addition towards using terms from the field names, we also added the ability for the tool's user to add keywords to the terms that define the field vector. This way the user who is exploring the schemas can help guide the tool to find more relevant matches.

After our evaluation of the tool, we concluded that information on field names alone is not sufficient to perform automatic schema matching and experimenting with using instances of the field was beyond the scope of what we wanted to achieve in this project. We believe our tool has the potential to be useful for individuals who are seeking to match metadata fields and link datasets but only after further improvements are made to the tool. Ultimately, we were forced to manually match the schemas of the resources, as the tool did not provide sufficiently relevant matches. We were fortunate that the fields we matched manually were relatively easy to find, however we were not able to manually traverse the entire Freebase collection, for example. Therefore, there remains the possibility that there is useful biographical information in Freebase that we were not able to use to enrich SNAC. For similar tasks in other projects, such a tool as we have developed here may well prove to be even more important as an aid for reviewing large, poorly organized corpora of metadata and schemas.

Chapter 4

Record Linkage

Automatically linking entity records is probably the most important challenge that needs to be addressed in any effort towards combining multiple Biographical collections. Unlike the schema matching problem discussed in the previous chapter, linking entities across multiple collections, or more precisely in our case from SNAC to Freebase and IMDB, should be done automatically. The sizes of the individual collections make it impossible for any manual, curation-laden effort to accomplish this.

This problem of automatically linking entity records, also called as the record linkage problem, has been an extensively researched topic. The standard approach towards this problem is to score a given pair of entities using a set of heuristics that determine the likelihood of the pair being a match. The most important of these heuristics is to compare the names of the entities using string matching techniques [4]. Additional information such as biographical facts, other than the names of the entities, may also be used for matching entities. However, since the SNAC dataset is rich only in names and existence dates - birth and death dates, we only use these two information facets in our approach.

The problem of matching entities in two different collections can be approached through both unsupervised or supervised machine learning techniques. Mann and Yarowsky[10] propose an unsupervised clustering technique where each entity is represented as a feature vector generated from a rich set of biographical facts that are automatically extracted from web query results pertaining to the entity. Cohen and Richman[5] propose an adaptive clustering technique where first a binary pairing function is learnt for potential pairs of entities. This is followed by a clustering step where entities are first represented as vertices in a pairing graph G. Edges in the graph represent a match between the entities and the edge weight is given by the confidence provided by the respective binary pairing function learnt for the entities; to find set of matching candidates, we find connected components in the pairing graph.

4.1 Method

The precise problem we wanted to solve in BioGraph is to automatically find entities in Freebase and IMDB that could be linked to corresponding entities in SNAC. Our approach was to cast this as a supervised learning problem and train a binary classifier for each entity in SNAC, similar to the pairing function proposed by Cohen et. al. Given a candidate entity from Freebase or IMDB, our binary classifier predicts whether the entity should be linked or not linked with the SNAC entity with some confidence. This approach was motivated by the fact that SNAC records are linked with name authority file records from the extensive VIAF collection and thus hold extensive information on alternate name representations and spellings for an entity, forming a potential source of information for training classifiers that learn varying forms of an entity's name. It has to be mentioned here that not all SNAC records are rich with information on alternate names, however most SNAC records of historic public figures, institutions and families do have this information associated with them. It is these records of historic information that we are motivated to enrich by linking with Freebase and IMDB. In addition to primarily using the name information for matching the entities in the three different collections, we also extensively use information on existence dates while matching person entities.

Our procedure for finding entity matches has essentially three steps as mentioned in listing 1. For each SNAC entity, we first find a potential list of candidate matches in Freebase and IMDB using our findCandidateMatches algorithm. This step, also known as blocking in record linkage literature, is important as it is computationally not feasible to run our binary classifier for each Freebase and IMDB entity, for instance this would amount to checking around 2M records for each SNAC record. Our findCandidateMatches algorithm dramatically reduces this number by using a set of simple heuristics, we would explain this algorithm later in this section. We then use our binary classifier to predict if each candidate entity is a potential match for the SNAC entity. For each candidate entity that is deemed as a positive match by our classifier, this step generates a *(SNAC entity, Candidate entity, score)* tuple, where the score represents goodness of the match. The final step clusters the tuples so that candidate entities are matched to SNAC entities with the best matching score. We first discuss our preprocessing and indexing strategies before going into the details of each of the steps.

LISTING-1 PROCEDURE FOR FINDING MATCHES BETWEEN SNAC ENTITY AND FREEBASE AND IMDB ENTITY

1. Find a set of match candidates using character shingles present in the names of the entities
2. For each matching candidate update matching score using existence dates information, if any.
3. Pick top-k of the candidate entities and compute likelihood of a match using a binary classifier trained on the SNAC entity.

4. Discard all candidate entities with less than threshold probability of a match.
5. Perform a postprocessing clustering step that associates each Freebase, IMDB entity to the SNAC entity with the highest matching probability.

4.1.1 Preprocessing

Using the rules generated in the schema matching step, we extract the following information for each entity in our three collections: name, type, gender, nationality, birth and death dates. In the case of IMDB, we collect only name, birth and death dates and inferred the nationality from the birth place information. Keeping aside the name field, an important challenge we had to deal with was the fact that each of the collections represent information for each of the fields in different ways. As an example, while SNAC uses country codes for encoding nationality information, Freebase uses full country names. Furthermore, the way in which the collections represent information for a field in not standardized. For instance in the case of SNAC, date information is specified both as just a year or as full date with month, day and year.

To handle this issue, we built custom procedures that normalized each field across the three collections. For the date field, we used the dateutil Python library[1] to parse dates that are represented in a variety of formats, we then extracted only the year as this information was prevalent in all the three collections. For the nationality field, we converted each country code in the SNAC record to a full country name using tabular lookup. This does not guarantee that the same country name will be used in all the three collections, however we handled this in our classifier where instead of checking for exact country name matches, we use string difference measures. Finally, normalizing the gender and type fields were straightforward as in all the three collections, we were aware of the finite number of values they take. We normalized entity names by first lowering the case and then by removing all digits and punctuation. We retained spaces in the names as space provides valuable information on the components of the name.

4.1.2 Indexing

Our goal was to create indexes that can enable us to quickly find candidate matches in Freebase and IMDB for a given SNAC entity based on name and existence date information. In this effort, we created three indexes for each collection: one index for names and two indexes for existence dates. Instead of indexing entities by individual name components, we indexed entities by overlapping character n-grams in the names. For example, for a name like 'Einstein Albert' the overlapping character tri-grams are ein, ins, nst, ...; we chose the tri-gram based index for our implementation. A downside to indexing by shingles rather than name components is the resulting large number of candidate matches, however, this approach is tolerant to variations in spelling.

[1] http://niemeyer.net/python-dateutil

To index person entities by birth and death dates, we first created date bins of size N years ranging from 1000 AD to 2000 AD, with an overlap of m years. For example with N = 5 and m = 3, the bins are 1000-1005, 1003-1008 and so on. We created two end bins for dates falling outside the above range and one additional bin for unavailable date information. The index for birth or death date then associates each year range to a set of entities. We chose N = 5 and m = 3 for our implementation. In addition to these indexes, we also created a master index where entities are indexed by unique identifiers provided by the individual collections.

4.1.3 Finding Candidate Entity Matches

The goal for the *findMatchCandidates* algorithm is to find the top-k entities in Freebase and IMDB that can be potentially linked with a given SNAC entity. Considering the enormity of the search space, wherein the potential candidate matches come from over 1.5M records, speed of the algorithm is crucial. We developed two simple criteria for a match candidate. First, the candidate entity should share at least three character n-gram shingle with the SNAC entity. Second, in the case of person records, the candidate match entity should share at least one date bin with the SNAC entity. Match candidates satisfying both these criteria can be quickly discovered using the indexes we built.

However, this approach has the potential to generate too many candidates. To reduce the space further, we first rank the match candidates by the number of shingles they share with the SNAC entity. We then increase the score of each match candidate depending on the date bins it shares with the SNAC entity. For instance, we give a higher score for a candidate that falls in the same birth and death date bin as that of the SNAC entity than a candidate that falls in only the same birth date bin as that of the SNAC entity. Once the candidates are ranked, we pick the top-k candidates with the highest score.

4.1.4 Supervised Learning for Predicting Entity Matches

Our goal was to design a binary classifier that given a SNAC entity and a candidate match entity from Freebase or IMDB, predicts whether the entity pair should be matched with each other. Our classifier first checks if the entity pairs are exact matches, if not it then estimates the probability of the entity pairs being a match using a supervised binary classifier that is trained to find matches for the SNAC entity. The most important factor we considered when designing the classifier was that the classifier should have very small false positive rate. In other words, we do not want our classifier to match two different entities. As would become evident from the following sections, this meant that we trained our classifier with a moderate bias towards negative samples. We also explain how we generated positive samples for a SNAC entity by randomly perturbing the information associated with it.

Cases of exact matches

Given an entity pair, the classifier first checks if they are exact matches. We consider two entities to be exact matches if at least one of their associated names exactly match with each other and they both share at least one birthDate bin or one deathDate bin. Essentially, we are looking for one exact name match and the match of atleast one existence date component. In the case of exact matches, the classifier returns a probability score of 1 for the entity pair.

A generative language model for entity names

Traditional record linkage approaches heavily use string distance functions for scoring entity pairs using variant names. Apart from using the standard string distance metrics, which we explain in the next section, we also experimented in building generative n-gram models for names using overlapping character shingle sequences obtained from the names associated with the entity. Our motivation was that given a number of variant names for an entity, it should be possible to train a probabilistic model that generates the names of the entities as a sequence of shingles.

The model can then be used to determine the probability of a test name being generated by the entity on whose names the model was trained. The fact that many SNAC entities have multiple variant names associated with them encouraged us to pursue this approach. Also, we felt that the language model approach would be better able to handle the cases where, for a pair of same entities, the components of the names are same but transposed in order, eg: "Albert Einstein" and "Einstein Albert". In the above example, if we wrap the two name strings by connecting the end of the string to its beginning, the resulting language model over a sequence of overlapping shingles should be able to capture the shingle sequences that are common to the names. For exactly matching entities with only their name components arranged in a different order, this technique perfectly predicts the exact match.

Before building the language model, we first preprocess the name in the same way as it was mentioned in the indexing section. However, we wrap the last character of the name and connect it back to the first character and compute all unique shingles across the wrap. Figure 10 illustrates this process for the name "Albert Einstein". Given a name n and its associated shingle sequences $s_1, s_2, s_{3,}...s_m$ where m is the length of the shingle sequence, the probability of the name being generated by the shingles is given by

$$p(n) = p(s_1)p(s_2|s_1)p(s_3|s_2, s_1)...p(s_m|s_{m-1}...s_1)$$

The above factorization can be simplified if we make certain assumptions on how a given shingle's occurrence depends on the previous shingles. In bigram models, we assume or approximate the factorization by assuming that the presence of the given shingle is dependent only on the immediately preceding shingle. That is in the case of bigram model, the above factorization can be written as

$$p(n) = p(s_1)p(s_2|s_1)p(s_3|s_2)...p(s_m|s_{m-1})$$

In general for k-gram shingles, we can write the factorization as

$$p(n) = \prod_{i=1}^{k} p(s_i|s_{i-1}s_{i-2}...s_{i-k+1})$$

The individual probabilities are computed as

$$p(s_i|s_{i-1}s_{i-2}...s_{i-k+1}) = \frac{c(s_{i-k+1}...s_{i-2}s_{i-1}s_i)}{\sum_{s_i} c(s_{i-k+1}...s_{i-2}s_{i-1}s_i)}$$

These individual probabilities form the parameters of the language model. For each SNAC entity, we training such language model over shingle sequences computed over all the variant names of the entity. For our experiments and final implementation, we chose shingle trigrams where k=3. Given a test name, its probability of being generated by the SNAC entity can be computed using the factorization mentioned above. However, there is a good chance that a test character shingle trigram under a given context will not be present in the shingle sequences used to train the model. To handle this situation, the probability estimates are usually smoothed. Discussion of the various smoothing techniques is beyond the scope of this report. For our experiments and final implementation we used Python's NLTK toolkit[2], the library uses the simple additive (or) lidstone smoothing and interpolates higher order models with lower order models. Given the small size of our sequences, this technique was adequate.

Features for matching entity pairs

For a given pair of person entities, we compute a set of features and construct a multi-dimensional vector. This vector is used as the feature vector to train our binary classifier. We used the following features for constructing the feature vector.

1. We compute the Jaccard similarity between the set of character shingles associated with each entity in the pair. If S is the set of shingles associated with the SNAC entity and T is the set of shingles associated with the candidate entity, the Jaccard similarity is defined as

$$score_{jaccard} = \frac{\| S \cap T \|}{\| S \cup T \|}$$

2. For each name associated with the candidate entity, we compute the probability of it

[2]http://www.nltk.org/

being generated by the language model trained on the SNAC entity names. We then take the average of the computed log probabilities.

3. For each pair of names associated with the SNAC entity and the candidate entity, we compute the Jaro-Winkler string similarity measure as mentioned in Cohen et al. We then take the average of the computed distance.

4. For each pair of entities, we compute the absolute difference between the birth (and death) dates. To penalize large deviations, we exponentiate the difference by 1.15. If D_1 is the birth date of the SNAC entity and D_2 is the birth date of the candidate entity, we compute the date difference score as,

$$score_{birthDate} = 1.15^{abs(D_1 - D_2)}$$

In cases when the dates are not available, we assume the difference in the dates to be that of the mean birthDate difference across the SNAC corpus. This was taken as seven years.

To expand our feature space, we computed the various interactions between the features by taking the products and sums of features with each other.

Generating samples for supervised learning

To train our binary classifier for each SNAC entity, we had to generate positive and negative training sample pairs for the entity. As mentioned earlier, we wanted our classifier to have low false positive rates, we therefore trained our classifier with 60% of training samples as negative and the remaining 40% as positive.

The negative samples were generated by pairing the SNAC entity with a random entity in the SNAC database. We used the alternate names of an entity to generate the remaining 40% of positive samples. Specifically, we generated a positive training sample by grouping three alternate names of an entity and then randomly perturbing the death and birth dates of the entity, such that the random perturbation didn't change the death and birth dates by more than 5 years. Additional positive samples were generated by randomly perturbing the names of the entity by replacing at most 3 random characters in the name and by randomly transposing the name components. For each entity we generated a minimum of 50 training samples in this manner. Entities with more alternate names were given a richer training set.

Binary classification using classification trees

Given the positive and negative training samples for a SNAC entity, we train a simple decision tree binary classifier using the rpart R library[3]. A sample tree learnt for physicist John Von Neumann is shown in Figure 1.

[3]http://www.stanford.edu/class/stats315b/minitech.pdf

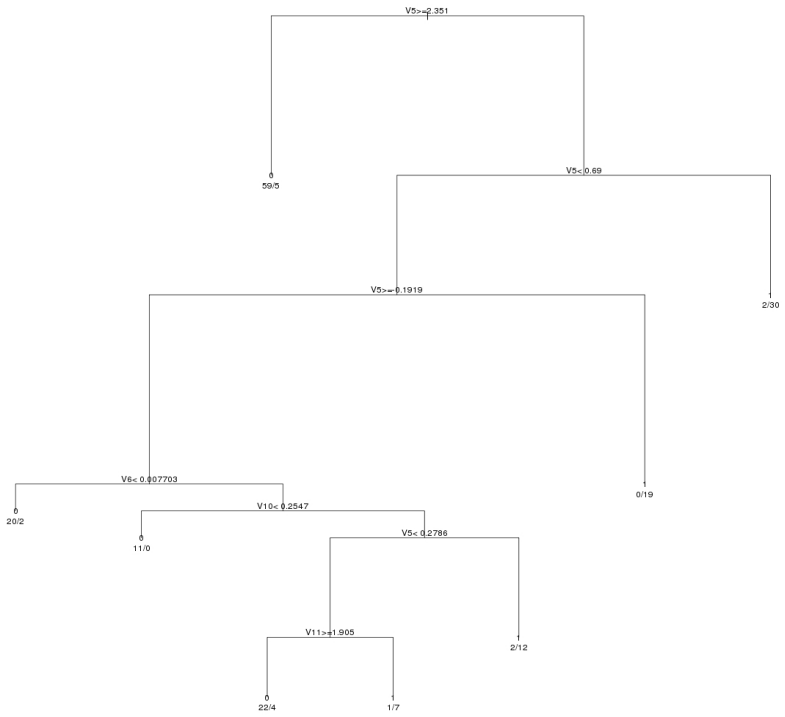

Figure 1 Example decision tree learnt for Von Neumann. The first branching of the three uses features derived from existence dates information. The tree then uses string distance metrics to decide on the classification result

We evaluated our classifier by splitting the generated samples into training and testing set in a 80%-20% split ratio. The following table shows the True Positive Rate and False Costive Rate obtained for three example entities, and for the entire corpus. In all the cases, the threshold probability for a match was set as 0.5.

Albert Einstein		George W Bush		Von Neumann	
TP: 78	FP: 11	TP: 39	FP: 9	TP: 182	FP: 14
FN: 25	TN: 145	FN:6	TN: 60	FN: 27	TN: 301
TPR: 75.7%		TPR: 86.6%		TPR: 87%	
FPR: 7%		FPR: 13%		FPR: 4%	

4.1.5 Postprocessing

For each SNAC entity, we first find candidate matches and then compute the feature vector for the candidate. We then score each candidates using our binary classifier. As mentioned earlier, exact matches are identified and given a score of one. For each SNAC entity, this step essentially generates a list of *(SNAC entity, match candidate and match score)* tuples. Once the matches for all the SNAC entities are evaluated in this manner, we match each candidate Freebase and IMDB entity with the SNAC entity that has the highest probability of being a match.

For considering a pair to be a match, we set a threshold probability of 0.85 in the case of Freebase and 0.9 in the case of IMDB. With these settings we linked 15,300 Freebase records with SNAC and 1100 IMDB records with SNAC. Given the sizes of the collection, we were not able to exhaustively evaluate the quality of these final matches. However, using the visualization tool, we were able to get a sense of how the algorithm performs for important public figures. We chose person records from SNAC's featured records[4] and evaluated whether our classifier matched these records in Freebase.

Bernstein, Leonard	Yes
Buffalo, Bill	Yes
Bush, Vannevar	Yes
Eames, Ray	No
Eisenhower, Dwight	Yes
Feynman, Richard Phillips	Yes
Fitzgerald Ella	Yes
Franklin Benjamin	Yes
Luce Clare Booth	Yes
Vuramontes, Helena Maria	Yes

The classifier didn't predict the match for *Eames Ray* as the Freebase entry does not have existence dates information. To ensure low false positive rates, we weighted the date feature relatively highly compared to features derived from entity's name. The classifier, as shown in the Von Neumann example, places great importance on date match. Even though the SNAC entity for *Eames Ray* had dates information, the fact the Freebase didn't have the information made the classifier to falsely classify the pairs as not a match.

[4] http://socialarchive.iath.virginia.edu/xtf/search

Chapter 5

Visualization

Biographical collections, including the ones we are concerned with in this project, have an enormous amount of latent social network information. The primary goal of the SNAC project, for instance, is to extract and combine the latent social network information from a huge corpus of archive context description records collected from multiple cultural institutions. Freebase explicitly encodes entity-entity network information through its quad store format . And finally, although IMDB does not explicitly encode entity-entity network information, this information can be easily extracted from the collection. In all the above-mentioned and similar such biographical collections, the primary means of access to the collection has been traditionally through an exploratory search interface (Figure 1).

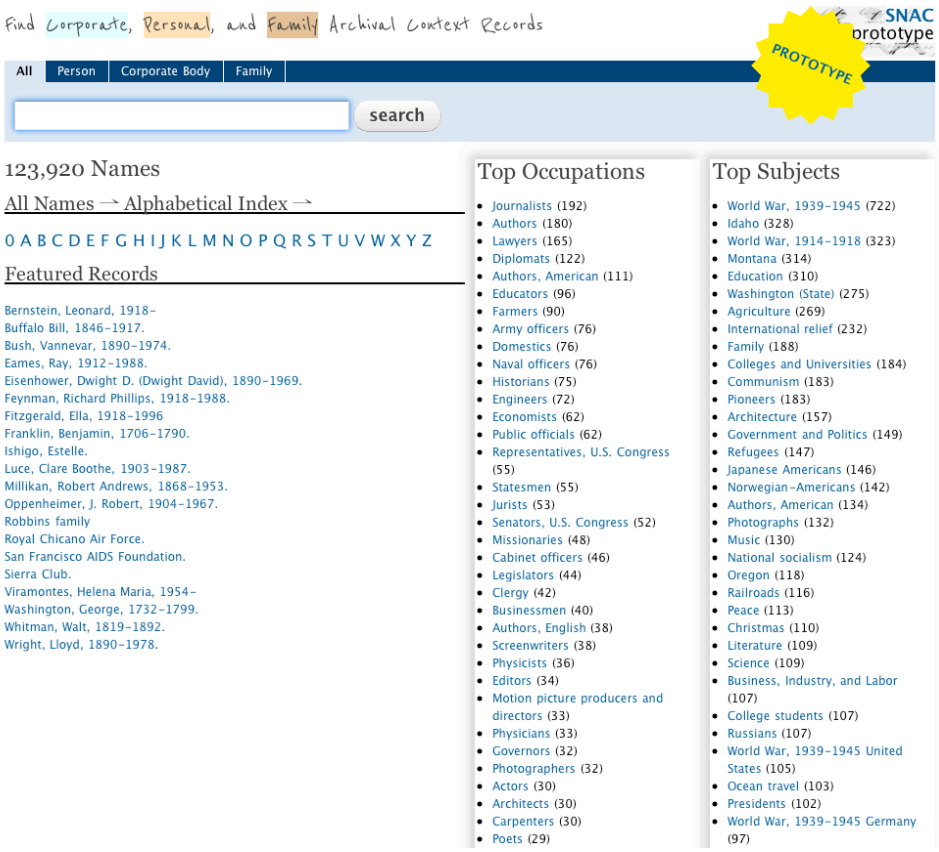

Figure 1 Exploratory search interface to browse the SNAC collection

Although such visual interfaces are ideal for finding information about single entities or entities grouped under facets, they are not suitable for exploring and discovering latent social network information about the entities. As an example, a historian might have the following question: “Find all physicists who were associated with both Richard Feynman and Von Neumann". To find answer to this question using the existing SNAC interface, the user has to first visit the pages of both Feynman and Von Neumann, then find connections to them and then manually distill the collected information to find the common connections. In general, faceted search interfaces are not ideal for finding social network related information, as the user has to go through multiple exploration and filtration steps to arrive at the answer. To make things worse, these exploration and filtration steps have to be often accompanied or followed by a post processing aggregation step as mentioned in the Feynman, Von Neumann example.

Other approaches for exploring social networks, or in general large directed multigraphs, include sophisticated query languages as in the case of Freebase’s MQL and graph exploration interfaces such as CGV[12] and Vizster[8]. While textual query languages can be powerful and flexible they require technical expertise from the user and this acts as a significant barrier towards their use. Graph visualization interfaces such as Vizster and CGV are specifically designed to find answers to network related queries. Such exploratory graph mining interfaces still require users to either start with the entire graph or a smaller subgraph and go through multiple exploratory and filtration steps to find the answer. In other words, users cannot directly express the structures they want to find as queries to the interface. Our motivation in BioGraph was to design a visual interface through which the user can intuitively express a network related query and directly find the answers with minimal or no exploration and filtration steps. This lead us to implement a query by example type interface where the user expresses the network structure he wants to find as a query graph. We implemented graph path search algorithms that take the query graph as an input and finds subgraph in the underlying graph database that exactly matches the patterns and structure of the query graph.

For instance, the Feynman and Von Neumann query example can be expressed as the query graph in Figure 2. The result from the graph path search algorithm is shown in Figure 3. By allowing the user to directly express the query as a graph, we have eliminated the need for multiple exploration and filtration steps that would have been required to arrive at the result graph. Our result visualization is exploratory; in the sense users can explore the result graph through zooming and linking features. Furthermore, our result graph layout algorithm uses information from the query graph to emphasize nodes in the result graph. We first discuss prior work that motivated us to follow this technique and then provide detailed discussions on our interface and methods.

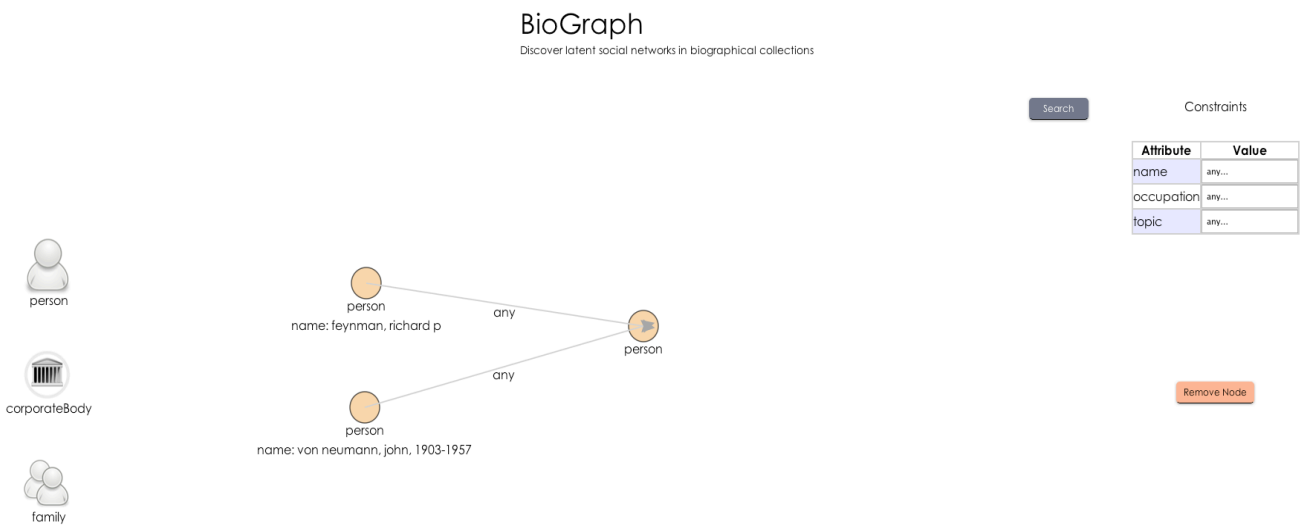

Figure 2 An example query graph to find common connections to Physicists Feynman and Von Neumann

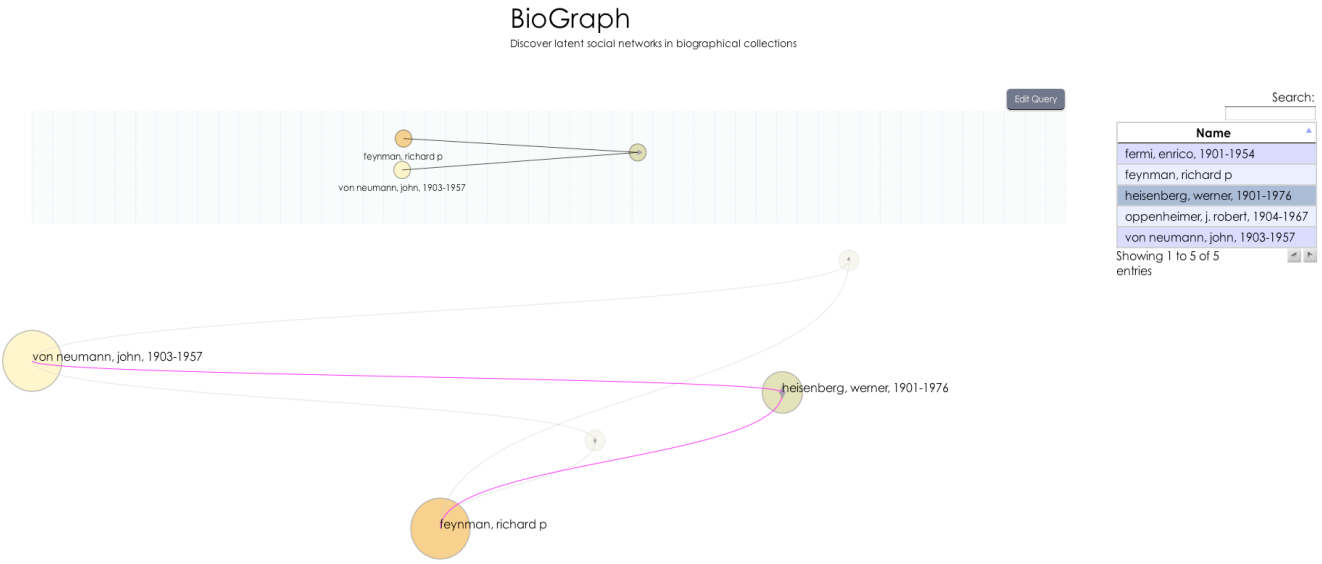

Figure 3 Result graph showing common connections to physicists Von Neumann and Richard Feynman as obtained for the query graph mentioned in Figure 2

5.1 Related Work

Query by example interfaces for visual graph exploration have been previously implemented in GRAPHITE [3] and QGRAPH [2] interfaces. The QGRAPH interface was specifically designed for both searching and manipulating knowledge repositories. QGRAPH returns only subgraphs that exactly match the user given query graph and supports complex queries;

query graphs can include inequality constraints, constraints on the number of edges to hop to find a pattern, etc. However, it is not clear from the work on how the user interacts with the interface while specifying the query. The GRAPHITE interface allows users to draw query graphs and finds both exactly and approximately matching subgraphs in the underlying graph database.

The interface seems to use the JUNG[1] library to layout the result graph. Interestingly, the result graphs presented in the work show strong structural correlation with the user given query graphs; we later argue this feature as important to have in query by example graph exploration interfaces. The work however does not discuss the layout of the result graphs and on how such structural correlation can be provided when the result graph is dense and is of moderate size with > 50 vertices. One of the key insights we obtained in this work is that visually depicting the correspondence between the query and the result graph is an interesting visualization challenge.

We extend both the related work through developing visualization and layout techniques that show visual correspondence between the user given query graph and the result graph. We argue that such correspondence and visual linking of the query graph with the result graph provides better access to the result visualization

5.2 Graph as query language

We gave considerable thought as to whether users would be comfortable in drawing graphs as queries and whether such a feature is required in the first place – is it possible to express the structural queries using purely menu driven interfaces? We feel that query graphs that try to find complex structural patterns, with multiple branches and cycles, are better expressed as graphs than as lists of menus. For applications where the target structures the user is interested in discovering are always going to be linear list of nodes, such as in the case time series, it would make great sense for the user interface to be more menu-driven. A good example can be seen in the work by Alan et. al for finding patterns in multivariate temporal data[1]. In that work, the user specifies the pattern he wants to find as a list of menus connected by edge patterns. In BioGraph, we provide a menu driven interface that allows the user to provide constraints to the edges and nodes in the query graph and we limit the number of vertex types to only entity types in our underlying biographical collection. For instance, we could have extended the vertex types to include Place and Time periods – this would have allowed the user to find two entities connected to the same place through a query graph. We felt increasing the number of vertex types would complicate the query graph interface. In general we believe designers of query by example interfaces for exploring graphs should think about what components of their graph database schema should be taken as vertex types and what components can be taken as attributes for those vertex types.

[1] http://jung.sourceforge.net/

5.3 Interface Components and Methods

Our visualization has three major components: A query graph interface that allows the user to draw query graphs based on the subgraph structure the user wants to find or explore. A subgraph search algorithm that finds the subgraph that exactly matches the query graph given by the user. And finally, a result graph visualization interface that visualizes the result graph and further provides zooming, linking and search features to further explore the result graph

5.3.1 Query Graph Interface

Our foremost design goal for the query graph interface (Figure 2) was that we wanted the user to spend minimal effort in drawing the graphs. First, we wanted to eliminate the need for the users to draw individual vertices and edges connecting the vertices. Second, we wanted the interface to guide the user towards drawing the query graph. We accomplished these two goals by linking the interface with the schema of the underlying graph database.

When the query graph interface initially loads, it also collects information about the schema of the underlying graph database; the interface then uses the schema information to guide the user towards drawing the query graphs. The left panel of the interface shows the different type of vertex types supported in the graph database. Clicking on a vertex type automatically places a query vertex of the vertex type in the query graph grid. Once there are two query vertices in the query graph grid, an edge can be drawn connecting each of them by first double clicking on the source vertex, followed by a double click on the destination vertex. Here again, the interface uses the schema information to prevent the user from drawing edges that are not supported by the vertex types. For example, if there are no outgoing edges from the family vertex type to the corporateBody vertex type in the underlying graph database, the interface prevents the user from drawing that edge.

An important feature of our interface is that users can annotate both the edges and vertices and add additional constraints to the subgraph (or) network structure they are interested in discovering. Clicking on any edge or vertex in the query grid would open a constraints panel. Because the interface is aware of the underlying schema, the constraints panel is tuned to each vertex and edge types and constraints pertaining to only those types are shown. Furthermore, each text boxes in the constraints panel have independent autocomplete features to allow the user easier access to the underlying database information. Once a constraint has been added, it is reflected in the query graph grid as an annotation under the corresponding vertex or edge.

5.3.2 Searching for Subgraphs

The subgraph search component of the interface performs the core function of searching for subgraphs in the underlying graph database that exactly match the user given query graph.

Before going into the details of the graph search algorithm, we first discuss our dataset and how we represent it as a graph.

Graph Representation and Indexing

To represent the graph structure of the collection in our database, we stored the entire collection in two tables, one for vertices and another for edges in an adjacency list format. It has to be mentioned that since we use a NoSQL database , we could store entities of more than one entity type in the same vertex table. For instance, the person entity has birthDate and deathDate information while this information will not be present for corporateBody entities. However, since NoSQL supports arbitrary column types, it was possible to store different entity types in the same table using a property field. This NoSQL feature also allowed us to develop simple procedures for matching query vertices rather than constructing complex SQL queries. A similar strategy was followed in creating the edges table, where each entity had a row associated with it with a column for each of the edge types. To aid our search algorithm, we created additional columns in the edge table that point to back edges. We indexed the property field of the vertex table and each of the edge type fields in the edge tables. Although we ended up creating multiple compound indexes, this was necessary to get quick search results.

Subgraph Search Algorithm

The subgraph search algorithm takes as input the query graph and produces a subgraph that exactly matches the vertex patterns and structure patterns as expressed in the query graph. A high level flow of the algorithm is listed in the listing 1. The given query graph is first traversed and a search path list of predecessor, successor query vertex pattern tuple is created. For each tuple in the search path list, the result graph is either extended or constrained. The result graph is extended if both the query graph vertices in the current tuple are unseen by our algorithm. The result graph is constrained if the search algorithm has already encountered either or both the query vertices. Depending on which of the query vertex is seen, the result graph is constrained using the result vertices previously obtained for that query vertex pattern. If at any stage any of the constraining procedures do not return a result, the graph search is aborted and a null graph is returned as the result of the search.

PROCEDURE subGraphSearch:

INPUT: The user query graph as a directed graph (V, E)

OUTPUT: The result subgraph as a directed graph(V, E)

1. Compute the search path: a list of (predecessor u, successor v, edge e) pattern tuples from the query graph
2. for each (u, v, e) pattern tuple in the search path:
   (a) do
   (b) if both the query vertices u and v are unseen
      i. extend the result subgraph by adding vertices that match vertex patterns defined in u and v and are connected by an edge matching the edge pattern e
   (c) if only u is seen
      i. constrain the result subgraph using result vertices corresponding to the query vertex u, using vertices corresponding to the vertex pattern associated with v and the edge pattern connecting the vertices
   (d) if only v is seen
      i. constrain the result subgraph using result vertices corresponding to the query vertex v, using vertices corresponding to the vertex pattern associated with u and the edge pattern connecting the vertices
   (e) if both u and v are seen
      i. constrain the result subgraph using result vertices corresponding to the query vertex u and v using the edge pattern connection the vertices
   (f) done

Both the extend and constrain procedures add vertices and edges according to the vertex and edge pattern tuple to the result graph. However, the constrain procedure removes vertices from the result graph: previously computed result graph vertices are removed if they do not satisfy the constraints defined in the tuple it operates on. The subgraph search algorithm naturally allows the user to provide disconnected query graphs. In which case, the result graph is obtained as a union of the results for the individual disconnected query graphs. See figure 4 for an example. It has to be noted here that though the results of the algorithm do not depend upon the order in which the query vertices are evaluated, the speed of the algorithm is tightly tied to the order. However, finding the most optimal order in which the query vertices should be traversed is beyond the scope of our project.

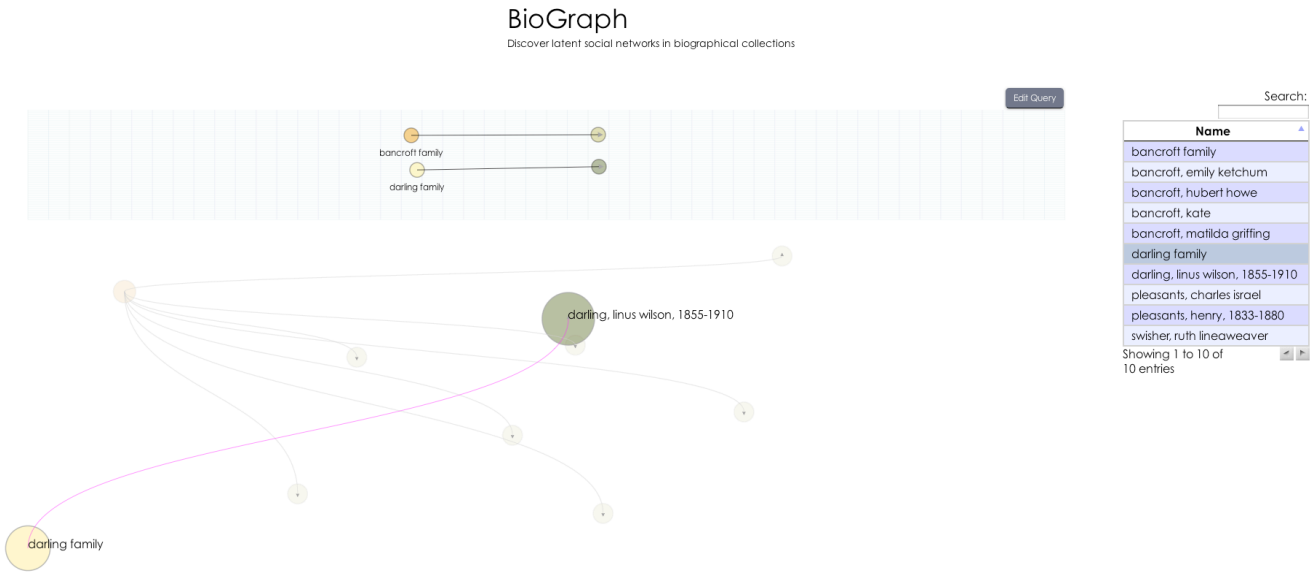

Figure 4 Users can provide multiple disconnected query graphs as queries to aid comparing different social structures. In this example, the query graph finds all persons connected to either the Bancroft family or the Darling family. The result graphs in this example are two disconnected subgraphs

5.3.3 Result Graph Visualization and Interface

Result graph interface

The Result Graph interface (Figure 3) has three components. The result graph obtained from the search process in displayed in the middle section of the screen. At the top of the screen, a scaled version of the query graph is shown. The right panel of the screen provides a text list of entities that are present in the result subgraph. All the three components are visually linked with each other. For instance, the user might search for an entity in the text list, clicking on any entity in the resulting search result list would highlight the corresponding vertex in the result graph. Also, clicking on the entity in the result graph would highlight the corresponding entity in the text result list. Finally, the vertices in the scaled miniature query graph are also linked with the corresponding vertices in the result graph. When a vertex in the result graph is clicked, that vertex and its connections are emphasized by a smooth animation. The emphasis is achieved by reducing the radius and increasing the opacity of the other vertices and edges not involved to the clicked vertex. A key component of the result graph interface is the result graph layout. Although the layout is primarily based on force directed spring layout, we use information from the query graph to appropriately emphasize and color code individual vertices in the result graph.

Result graph layout

An important feature that is expected in Search User Interfaces is that the result visualization should provide some form of query related feedback to the user [7]. For instance, text search interfaces such as general search engines provide search result snippets for each result document where the snippet highlights search terms in the result document. The snippet provides some clues to the user as to why the document was retrieved by the search engine. We wanted to provide a similar functionality in our result visualization. We wanted our result graph layout to provide feedback on why each of the vertices and edges were returned. We have come up with naive approaches towards this purpose using clues from the query graph. First, each vertex in the result graph is encoded with the same color as that of the query vertex that is responsible for generating it. This linking allows the user to immediately associate a vertex in the result graph with the corresponding vertex in the query graph.

However, it has to be mentioned that there are certain cases when such unique mapping of result vertices to query vertices may not be possible, For example, consider the query and the result graph in Figure 5. The query is to find all cabinet officers connected to Abraham Lincoln, who in turn had connections from Lawyers. Because there were few Lawyers who were also Cabinet Officers, it was not possible for the layout to associate, cleanly, each result vertex to the corresponding query vertex. A naïve solution to the problem would be to color code the vertices based upon the current context provided by the vertex that is being focused. In the above example, the lawyers connected to the cabinet officer Lincoln Robert, Todd could be given the green color (associated for lawyers) as the current focus is on Lincoln Robert. However, this would result in changing color encodings for the vertices and detailed user study is required to find if the solution is acceptable. It should be noted here that though Abraham Lincoln was also a Lawyer, the result layout assigned the right color to the corresponding vertex, as that query vertex was considered 'important' by the layout algorithm and the associated color was left unchanged.

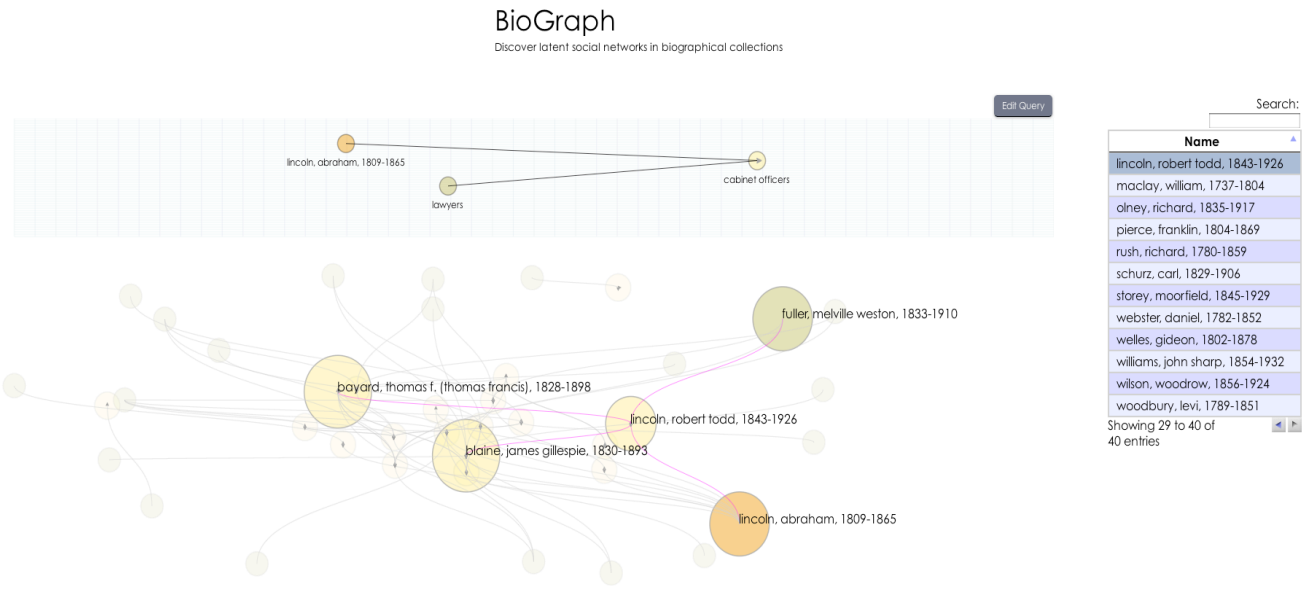

Figure 5 The figure shows query and result graphs corresponding to the social structure involving Abraham Lincoln, Cabinet Officers connected to him and Lawyers connected to them. In this figure, the vertex corresponding to Lincoln, Robert Todd is focused and in and out connections from the node are highlighted. Both Baynard Thomas and Blaine, James Gillespie are cabinet officers connected to Abraham Lincoln, however, as Lawyers they are connected to Robert Todd Lincoln. Thus creating an ambiguity on which color to associate for that particular node.

The result layout algorithm considers query vertices as important depending on the constraints associated to them by the user. We use the following simple heuristics to associate importance scores for query vertices:

1. End query vertices are more important than query vertices in the middle.

2. A query vertex with no constraints is less important than a query vertex with more constraints. For each constraint we add points to the vertex and sum the total number of points.

3. A query vertex with a name constraint is more important than a query vertex with an occupation constraint as the latter will yield a set of vertices while the former would yield one specific result vertex. Each result vertex is given importance score points depending on which query vertex generated them and the scores are normalized across result vertices

While laying out the result graph, vertices with higher importance scores are drawn with circles of relatively bigger size than the vertices with lower importance scores. A simple simulated annealing algorithm [9] as defined in listing 2 determines the size of the circles.

PROCEDURE simulatedAnnealing:

INPUT: a list of result graph vertices and their associated radius

OUTPUT: a list of result graph vertices with their radius optimized using their importance scores and density of the graph

1. E <- scoreLayout()
2. while (not Termination Condition)
   (a) do
   (b) perturbLayout()
   (c) newE <- scoreLayout()
   (d) if ((newE > E) and (Random() < (1.0 – $e^{-\triangle T/T}$)))
      i. revertLayout()
   (e) else
      i. E = newE
   (f) decrease T
   (g) done

We first compute the position of the result vertices using a force directed spring layout algorithm using the NetworkX Python graph library[6], with number of iterations as 15. We then assign a constant radius of 2px to each result vertex. We take this as the initial layout for the simulated annealing algorithm. Each perturb step of the algorithm increases the result vertices by a percentage that is proportional to its importance score. The score step of the algorithm finds out how many result vertices overlap each other, this is taken as the error score. Vertex overlap was quickly computed by approximately hashing the vertices to 100 square regions in the screen. The goal of the algorithm is to thus find the optimal radius that should be assigned to the result vertices so that there are minimal overlaps.

5.4 Evaluation

The query response time in BioGraph is in the order of milliseconds thanks to the extensive bidirectional indexing. However, certain queries can take longer times – especially for query graphs with generic person entity nodes that have only outgoing edges. As mentioned earlier, query response times can be further optimized if the optimal order of evaluating the query vertex pattern tuples could be determined. We could not perform any user study

given the limited time and our focus on other components of the project. We however got feedback from two users who tried using the interface. Both the users, who are professional computer programmers, found it difficult to express correctly the Lincoln query (Figure 5). However after spending 10-15 minutes with the interface, they were comfortable in expressing complex queries such as: "find all physicists with connections to Caltech who were in turn connected to persons with atomic bomb as a topic". It was also interesting to note that the first query that both the users tried was to find connections between two person entities with no constraints – essentially they were trying to visualize the entire graph.

Further user evaluation is required to find out if users are more comfortable starting with the entire graph and drilling down BioGraph will be user evaluated with 8 users with mixed technical background. The goal of the user study is to find out if users of varying background are capable of expressing complex queries as graphs. Towards this purpose, we have designed a set of 8 query exercises with known answers. Users will be given an initial demo and allowed to play with the interface for 10-15 minutes, they will then be asked to find answers to the 8 exercise questions. We hope this study would throw some light on whether graphs as query languages are an useful approach to pursue and also on the general effectiveness of the interface.

Chapter 6

Conclusion

We have described BioGraph, our efforts to make the SNAC collection more accessible by developing tools for linking the collection with online, crowdsourced resources such as Freebase and IMDB and by developing an interface that, we believe, is more ideal towards discovering latent social structures in the SNAC collection.

Towards linking the SNAC collection with Freebase and IMDB, and in general any biographical collection, we designed a binary classifier based on decision trees. A highlight of our implementation is building and using language models, built over overlapping shingle sequences obtained from a name, towards comparing names. Our hypothesis was that such a language model handles the cases when two name strings differ only by the order of arrangement of their name components. Based on a small test evaluation sample, the language model in conjunction with the decision tree based classifier, trained for each SNAC entity, worked well towards linking our collection with Freebase and IMDB. Although, we were able to match only 15, 300 SNAC entities with Freebase, and 1100 SNAC entities with IMDB, we feel that the number matches can be increased by increasing the cut-off probability for the decision tree classifier. However, this would result in a higher false positive rate. In the future, we would like to experiment alternate strategies such as using words, derived from other biographical facts, towards building features for matching the entities.

In BioGraph's visual interface, users express the network structure they wish to discover in the underlying social graph database as a query graph. A subgraph search algorithm takes the query graph as the input and searches for subgraphs in the underlying social graph database that exactly match the vertex and edge patterns provided in the query graph. BioGraph extends previous work in query by example graph exploration interface by introducing techniques to layout the result graph such that there is a visual correspondence between the user given query graph and the result graph. Currently, BioGraph emphasizes important vertices in the result graph by adjusting the relative sizes of vertices in the result graph and by color encoding the vertices according to the query vertices that generated them. Many extensions are possible here: the position of the vertices or entire connected components in the result graph can be laid out based on the user given query graph. This would be of great value to the users when they express multiple disjoint query structures in the query graph.

It will be interesting to incorporate approximate subgraph search algorithms in BioGraph. However, we feel there are significant challenges in visualizing result graphs that only approximately match the user given query graph. As evident from our limited user study, we found that users find it difficult to understand result graphs with more than 50 vertices, even if they exactly match their query graphs. Approximate matches would only aggravate this issue. In general we believe that future such interfaces should focus on visualization techniques that help users better understand the reason behind the system retrieving a result graph for a query. Although BioGraph's interface was designed to search social networks, our techniques, strategies and arguments are applicable for any query by example interface for exploring graphs.

Bibliography

[1] Jerry Alan, Fails Amy, Karlson Layla, and Shahamat Ben Shneiderman. A Visual Interface for Multivariate Temporal Data: Finding Patterns of Events across Multiple Histories.

[2] H. Blau, N. Immerman, and D. Jensen. A Visual Query Language for Relational Knowledge Discovery. Technical report, 2001.

[3] Duen Horng Chau, Christos Faloutsos, Hanghang Tong, Jason I. Hong, Brian Gallagher, and Tina Eliassi-rad. GRAPHITE: A Visual Query System for Large Graphs.

[4] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. pages 73–78, 2003.

[5] William W. Cohen and Jacob Richman. *Learning to match and cluster large high-dimensional data sets for data integration*. ACM Press, New York, New York, USA, July 2002.

[6] Aric Hagberg, Dan Schult, and Pieter Swart. NetworkX Reference. page 331, 2010.

[7] M Hearst. *Search User Interfaces*. Cambridge University Press, 2009.

[8] J Heer and D Boyd. Vizster: visualizing online social networks. *IEEE Symposium on Information Visualization 2005 INFOVIS 2005*, 5(page5):32–39, 2003.

[9] S Kirkpatrick, C D Gelatt, and M P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[10] Gideon S Mann and David Yarowsky. Unsupervised personal name disambiguation. In *Proceedings of the seventh conference on Natural language learning at HLTNAACL 2003*, volume pages, pages 33–40. Association for Computational Linguistics, 2003.

[11] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, December 2001.

[12] Christian Tominski, James Abello, and Heidrun Schumann. CGVâAn interactive graph visualization system. *Computers & Graphics*, 33(6):660–678, December 2009.