Identifying Efficient Learning Paths

Sachinrao Panemangalore, Andrew Simpson, Rajkiran Veluri

Introduction

Search engines provide good results based on keywords; however, these results do not form a coherent path of information that can be used to examine further depth of a topic. We propose to use a form of information cartography to find the most efficient learning paths. These paths may include branches to subtopics and may also intersect at common topics. The maps produced are similar to city metro lines where each station conceptually related to the previous and next station where further stations offer more depth and branching into new ideas. Current implementations of these maps, known as metro-maps, have subjective final outputs, which hinders optimization. A modified version of PageRank will be discussed that may be useful in finding an objective way to score these maps. Additionally, these metro maps offer more that just a high level overview of relations. These maps can lead to information discovery by finding the contents nodes (clusters) that have intersecting terms. This may be using in areas such as discovering what skills are needed for a job to discovering new areas of scientific research. Term intersection will be examined in the context of job skills related to companies.

Prior Work

Information Cartography: Creating Zoomable Large-Scale Maps of Information (Dafna Shahaf, et al.) [4]

Summary

In the paper a method is presented to create zoomable maps of information. The motivation behind this paper is that some users may want a high level overview of a subject whereas others want a high level of detail. Previous models of metro-maps were of fixed resolution, and they were also of limited scale. Older version can only be computed on a few hundred documents. This implementation can handle hundreds of thousands of documents. The documents examined were news articles with an associated timestamp.

The basic implementation of the algorithm is as follows:

- Find the optimal structure, of a metro-map M^*
  - Find an initial set of clusters C optimizing C-qual(C)
  - Find an initial set of lines Π optimizing S-$qual(\Pi|C)$
  - Perform local search to improve lines without sacrificing cluster quality
- Find a subset map $|M_k^*| \leq K$ maximizing coverage

Finding good clusters (C-qual(C)) were by found creating a word co-occurrence graph, a graph where every word is a node and the weighted edges are how often a word occurs with another word in the set of documents D, for each time-step. A fast overlapping community detection method for undirected networks was used (BigClam [5]) to discover the communities in the co-occurrence graph. In order to reduce noise, the top 50 tf-idf words for each document were used for creating co-occurrence graphs. Coherence is the next important property (described in [1]). Coherence is a measure of the similarity between each pair of consecutive clusters along a line. The calculation of coherence favors a small set of words and smooth transition between nodes. An important property of coherence (found in [1]) is that it is a global property, which means the optimum coherence cannot be deduced through local interactions. There are many possible coherent lines formed. The structural cost metric (S-$qual(\Pi|C)$) was used to find the best line. This metric is the cost of a line plus words that did is not in the next node. Finally, the maximum number of lines is limited to K. The limiting of the lines is done by a greedy algorithm that maximizes coverage.

Judging the quality of the results is subjective to the human user. In order to find the quality of the metro-maps produced the Mechanical Turk service was used. These users judged the produced metro-maps along with Google

search engine results, KeyGraph (a graph based topic detection solution), and previous versions of metro-maps. The users preferred these new maps to Google 72%, KeyGraph 65%, and older metro-maps 58% percent of the time.

Shahaf, Dafna, Carlos Guestrin, and Eric Horvitz. "Metro maps of science." Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2012.) [2]

Summary

The authors propose metro-maps as a method to solve the problem of information overload arising from the proliferation of scientific literature. As the number of publications increases, a fine-grained approach to track the main developments in a field of study is required, especially for new entrants to the field such as interdisciplinary researchers or graduate students introduced to the field. Academic search engines and information retrieval tools do not provide a big picture of what is happening in the field, its complexity and interconnections. Metro-maps help by organizing this information in the form of a graph where nodes represent research papers and edges represent citations to other papers. Then coherence, coverage and connectivity of this graph is measured. This paper mainly touches upon class topics related to network modeling and topology like link prediction and network inference.

A metro-map is a directed graph (G, Π) where Π are paths known as metro-lines. Each edge must be part of a metro-line. Also, following a path along a metro-line must lead us through a coherent understanding of how a story or concept evolves. For scientific papers, coherence stems from direct influence, which is defined as the probability of a direct path between two papers p_i and p_j. Another type of influence is ancestral influence which is the probability that p_i and p_j have a common ancestor. These two types of influences determine coherence between nodes of a metro-line.Also, the strength of a chain (a set of nodes in a metro-line) is measured as the strength of its weakest link, so it is desirable to have chains that have at least a minimum strength τ. Shorter chains are more desirable than longer chains, as with increasing length, the influence of a node on another node decreases. An algorithm to find short chains (of length m) is given in [3].Divide and conquer approach is employed to find short candidate chains and join them and Monte-Carlo sampling is done to limit the number of samples to an order logarithmic to the number of papers, since the problem is NP-complete. Coverage, $cover_p(q)$ is defined as the probability that a random walk from paper q to its ancestors reaches p before termination, taking into account all the copies of p. A copy of p is an instance where p occurs in the graph G. With this notion of coverage, the coverage of a map M is defined as the weighted sum of the coverage of each paper q: $Cover(M) = \sum_q \lambda_q cover_M(q)$. $cover_M(q)$ can have maximum value 1, so adding a paper when the coverage for q is close to 1 provide little value in terms of coverage. Hence, to maximize coverage, we pick papers that show diversity. We want coverage such that the metro-map covers most of the important features of the corpus. It is ideal to have maximal connectivity, or connections between all pairs of paths in the metro-map but connections between lines are chosen in a way that does not reduce coverage.

A joint objective function is then computed to find a metro map M such that connectivity, $Conn(M)$ is maximized, given that the coherence, $Coherence(M) \geq \tau$ and coverage, $Coverage(M) \geq (1 - p\epsilon)\kappa$, where ϵ is a pre-determined constant and κ is the maximal coverage over all maps with $coherence \geq \tau$.

To judge the quality of the metro-map, an experiment was performed where subjects were given access to Google Scholar with the metro-map (GS+MP) and other subjects were given only Google Scholar (GS) and asked to identify seminal papers in 5 research directions in reinforcement learning, and it was observed that precision of GS+MP subjects was greater than GS subjects by 9-10% and recall improved by 27-30% with the use of Metro-maps.

Overlapping Community Detection at Scale: A Nonnegative Matrix Factorization Approach [5]

Summary

This paper presents a highly scalable method to detect communities in large networks. The motivation behind this paper is that communities often represent organization structures in networks. They represent disciplines in citation networks. Detection of these communities may represent new domains for research and specialization. The paper proposes an algorithm called BigClam. The presented algorithm has been evaluated in the paper on a range of network sizes and structure. This includes 4 online social networks, DBLP citation network and the Amazon product co-purchasing network. The basic description of the BigClam generative model for networks is as follows:

Simplifying Assumption: Each network is represented as an unweighted, undirected static graph. Each connected component of the group is considered a separate ground-truth community.

Data

Data Collection Process

Wikipedia offers a complete dump of all articles in XML format. This XML file was parsed into plain text and the top 500 TF-IDF values were calculated for each page. The dump also contains SQL files that contain metadata about pages, page links, category links. The categories appear to not have a hierarchical structure, but provide information related about pages. The page SQL file contains the integer ID, page title, and other less relevant metadata. The page links SQL file contains all the links to and from pages.

Dataset Information

Many Wikipedia pages contain links to pages that do not exist (single digit percentage). The total number of articles is over 5 million. The average out degree is 229. The dataset follows a power law distribution with an α of 2.4. In the Wikipedia dataset articles can be reached relatively quickly from another article, which means it has small world properties.

Wikipedia Subset Information

The entire Wikipedia dataset is too large to be used with the version of MetroMaps used. A subset was constructed with over 500 articles starting at the Java programming language article and expanding out with a breadth first manner. Each step in the expansion was used as a time stamp for the MetroMaps algorithm. Due to the small world effect, can reach any page in a short number of hops, the pages were limited to categories related to computing and math. The pages with the highest centrality are all related to programming languages. This is not unexpected due to the category limitations. Interestingly, Java itself is not the top position (position 5 in both types of betweenness). Basic information about the subset is listed in table 1 and table 2. The top interesting communities of the subset are shown table 3.

Average Degree	Average Clustering Coefficient	Nodes	Edges	Distribution
1.95	0.000650	513	1002	Power Law

Table 1: Basic network information for the Wikipedia subset.

	1st	2nd	3rd
Betweenness Centrality	(C_Sharp, 20849)	(Dart, 12615)	(Clojure, 9585)
Closeness Centrality	(C_Sharp, 0.16)	(Dart, 0.14)	(Clojure, 0.13)

Table 2: Betweenness and closeness centrality of the Wikipedia subset.

Group	**Size**
Computer languages	46
Computer languages (C like)	42
Frameworks (mostly JVM related)	41
Front end technologies	39
P2P applications	31
Statistics and related applications	24

Table 3: Top interesting communities using the Girvan-Newman algorithm for community detection.

Mathematical Background

Linear Programing

Linear programing is a subset of convex optimization methods. The basic idea is to optimize a function subject to constraints. The constraints and function are linear. There are several ways to solve these optimization problems.

The standard way is to use the simplex algorithm.

Metromaps Algorithm

The basic implementation of the algorithm for the state-of-the-art version of MetroMaps (listed in the prior works section) is as follows:

- Find the optimal structure, of a metro-map M^*
  - Find an initial set of clusters C optimizing C-qual(C)
  - Find an initial set of lines Π optimizing S-$qual(\Pi|C)$
  - Perform local search to improve lines without sacrificing cluster quality
- Find a subset map $|M_k^*| \leq K$ maximizing coverage

The specifics of each subcomponent will be discussed in the following subsections.

Term Frequency Inverse Document Frequency

TF-IDF stands for term frequency inverse document frequency. In the MetroMaps algorithm TF-IDF can be used to weight edges between documents and words. TF-IDF give more weight to rare words and less to common ones. For example the word "the" is almost in every English document whereas a word like "tensor" may be very important. Knowing a document contains "the" tells essentially nothing about the contents of that document. Additionally, documents vary in length. The word "president" may appear more times in a long text book unrelated to anything presidential whereas a short article may be exclusively about the president. The basic formulation of TF-IDF is in equation 1 where n_t is the number of a particular term t in document d where N_d is the number of terms in document d. N is the total number of documents and N_t is the number of documents with term t in them.

$$\begin{aligned} TF(t) &= \frac{n_t}{N_d} \\ IDF(t) &= log(\frac{N}{N_t}) \\ TFIDF(t) &= TF(t)IDF(T) \end{aligned} \tag{1}$$

BigClam and Cluster Quality Scoring

BigClam is used for the quality scoring metric for the metromaps algorithm. BigClam is used to detect communities at large scale with a nonnegative matrix factorization approach. Previous methods of community detection lacked scale and also had a flaw in the assumption that people in overlapping communities were less likely to be connected. The basic idea is to find the maximum likelihood of a bipartite community affiliation graph given a network. The specifics of the algorithm are listed below.

1. The algorithm represents node community memberships with a bipartite affiliation network that links nodes of the social network to communities that they belong to.
2. Each affiliation edge in the bipartite affiliation network has a nonnegative weight. The higher the nodes weight of the affiliation to the community the more likely is the node to be connected to other members in the community.
3. For each community a pair of nodes shares we get an independent chance of connecting the nodes. Thus, naturally, the more communities a pair of nodes shares, the higher the probability of being connected.
4. Let F be a nonnegative matrix where F_{uc} is a weight between node $u \in V$ and community c Given F, the BigClam generates a graph $G(V, E)$ by creating edge (u, v) between a pair of nodes u, v with probability p(u, v): $p(u, v) = 1 - e^{-F_{uA} \cdot F_{vB}}$ where F_u is a weight vector for node $u(F_u = F_{u\cdot})$.
5. BigClam allows for rich modeling of network communities which are (a) non-overlapping, (b) overlapping, (c) nested.

6. Community Detection: BigClam poses the community detection problem as a non-negative matrix factorization problem. Given an unlabeled undirected network $G(V, E)$, the algorithm aims to detect K communities by fitting the BigCLAM (i.e., finding the most likely affiliation factor matrix to the underlying network G by maximizing the likelihood $l(F) = logP(G|F)$ of the underlying G: $l(F) = \arg\max_{\mathbf{F} \geq 0} l(F)$ where $l(F) = \sum_{(u,v) \in E} log(1 - exp(-F_u F_v^T)) - \sum_{(u,v) \notin E} F_u F_v^T$.

Influence

Influence is a value that represents how much a document d_i has on a document d_j with respect to word w. How can this be measured? If words are documents are set up as a bipartite graph (edges between documents and words), with the weights on the edges being the TF-IDF scores for the word in each document (normalized over the sum of the TF-IDF scores of each document for a word), then finding the steady state probability of being at a node can be found. This is the same as personalized PageRank. The steady state distribution ($\Pi_i(v)$) can be measured as $\Pi_i(v) = \epsilon \mathbb{1}(v = d_i) + (1 - \epsilon) \sum_{(u,v) \in E} \Pi_i(u) P(v|u)$ where $P(v|u)$ is the probability of reach v from u (document at time or other demarcation i to document at $i+1$). To find the influence of a word, w, the bipartite graph is modified to turn that word, w, in to a sink. This means essentially means the word is deleted from the document at $i + 1$. The steady state probability is then recalculated. The first steady state probability is subtracted from the new one with the sink ($\Pi_i(d_j) - \Pi_i^w(d_j)$). This is the influence value.

Coherence

What is a good metric for a coherent story line. An initial idea might be find the shortest path between a starting and an ending document. This produces stories lines that are not logically coherent. A story line is only as strong as its weakest link as some non-related stories may score high. Also, words may be important to an article even if they are not in that article. Jitteriness can be an issue where the story line jumps back and forth. Also, too many words active per transition makes for a hard to follow story. The goal for the coherence metric is to maximize equation 2 .

$$Coherence(d_1, ..., d_n) = \min_{activations} \min_{i=1...n-1} \sum Influence(di, d_{i+1}|w) \mathbb{1}(w \text{ active in } d_i, di + 1) \tag{2}$$

How can equation 2 be maximized subject to these constraints? The optimal solution can be found through linear programing (a subset of convex optimization). This makes coherence a global property that cannot be found through only local transitions (discovered in [1]). The optimal solution can be found through linear programing (see [1] for the full linear programming setup). The implementation has a complexity of $O(|D|^2|W|)$, which will not scale well. For each line, a limited number of documents can be used that have a high probability during a random walk (essentially PageRank) this addition gets the complexity down to a more manageable $O(|D||W|)$.

Structure Cost

How can line quality be determined? The idea is to maximize the number of words that come across the transition while minimizing the total number of lines. The cost of a set of lines Π given a set of community clusters C (found by BigClam). A line $W(\pi)$ is the line that covers a set of words.

$$Scost(\Pi|C) = \sum_{c \in C} |c \setminus \bigcup_{\pi c \in \pi} W(\pi)| + \sum_{\pi \in \Pi} |W(\pi)| \tag{3}$$

Candidate Lines

After find a set of word clusters C during each time step, the goal is to find a set of candidate lines. The idea is to start with a bipartite network $B(C, W, E)$ where C are the word clusters, W are the words, and E are the edges between them. The algorithm is a modification of BigClam. Again, maximum likelihood is used to the most likely weights. Word $w \in C$ has a membership weight ofF_{wg} for group g. Cluster $c \in C$ has a membership weight H_{cg} for group g.

$$\begin{aligned} \hat{F}, \hat{H} &= \operatorname*{argmax}_{F,H \geq 0} l(F, H) \\ l(F, H) &= \sum_{(w,c) \in E} log(1 - \exp(-F_w H_c^T)) - \sum_{(w,c) \notin E} F_w H_c^T \end{aligned} \tag{4}$$

Good Structure/Connectivity

After finding a set of candidate lines, the structure cost needs to be minimized. To minimize this directly is computationally costly; however, $Squal(\Pi|C) = -Scost(\Pi|C)$. This is a submodular function (adding a new line adds cost, but may have a smaller number of words covered). Equation 5 is maximized to find the best lines ($|W(\pi)|$ is the number of words covered by a line).

$$Squal(\Pi|C) = -Scost(\Pi|C)$$
$$2w_c - \sum_{c \in C} |c \setminus \bigcup_{\pi c \in \pi} W(\pi)| - min(\sum_{\pi \in \Pi} |W(\pi)|, w_c) \tag{5}$$

Coverage and Budget Constraints

After finding a good structure (the map), the map can be scaled. The way to do this is to use the greedy hill climbing algorithm on equation 6 (π are the lines and M is the map). This is a near identical relative to the greedy hill climbing algorithm for influence maximization, and has the same guarantee of $(1 - \frac{1}{e})$ times the optimal solution. The algorithm examines the importance of each line and can remove the ones that have the least coverage.

$$IncCover(\pi|M) = Cover(\pi \bigcup M) - Cover(M) \tag{6}$$

Results and Findings

Code Base and Data

We used the https://github.com/snap-stanford/MetroMaps implementation as our baseline of the metro-maps algorithm. This code has a white listing concept. These whitelists contain terms that should be in the clusters produced. We updated the text pre-processing logic in the code to use Python NLTK for preprocessing.

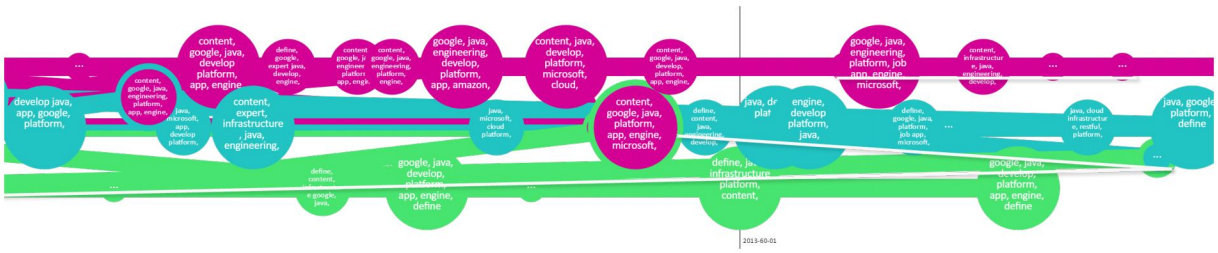

Figure 1: Sample output from the existing code base. The clusters follow a Wikipedia breadth first expansion using a whitelist of software engineering terms. Each line follows a concept.

PageRank and Information Density

One of the main draw backs to the metromaps algorithm is that the quality of the final output is subjective to human judgment. Parameters have to be manually tweaked in order to produce the best map. A metric that can be calculated on the final map would be helpful as the parameters that were used to create the map could be optimized through machine learning. An objective way to do this was tested using a modified version of PageRank.

PageRank is essentially the steady state probability that a random surfer of the network will be at any node at any given time. There are a few problems to the basic PageRank algorithm, such as spider traps and dead ends, that cause the final form to be $v = \beta M v + (1 - \beta)\frac{e}{n}$ where v is the PageRank vector, M is the adjacency matrix with the values of the columns summing to one (if all zeros replace with $\frac{1}{n}$), β is the tax for teleportation (avoids spider traps, etc.), and $(1 - \beta)\frac{e}{n}$ adds in the chance to jump to any node (e is a vector of all ones). This set up run until the values of v converge to the steady state probabilities of being at a node.

In order to examine PageRank with metro-maps, a mini-network for each cluster was constructed. These mini-networks consisted of the nodes in the cluster in question along with the nodes in neighboring clusters. In order to differentiate between the primary cluster and its neighbors, personalized PageRank was used. This is a slight modification from the standard PageRank algorithm in that if a node were to teleport it would be back to one of

the nodes in the primary cluster and not any node in the neighbors. This logic places the surfer as a resident of the primary cluster and a tourist to the other clusters. The primary cluster would be the surfer's home city and the other links would be outbound highways. A notion of city-to-highway ratio was examined. Different quality maps were constructed by tweaking parameters such as line number Jaccard similarity of cluster, etc. In figure 2 the results were not helpful in finding a good metric to use for map optimization; however, if the values are scales to take into account the size of of the "city" and the number of neighbors, the results look much better (figure 3 see equation 7 for score scaling). Other features were examined (total cities, total size, etc.), but did not provide any metric of value.

$$quality = \frac{1}{N} \sum \frac{|v_{city}| \sum v_{city}}{|clusters_{neighbors}| \sum v_{highway}} \tag{7}$$

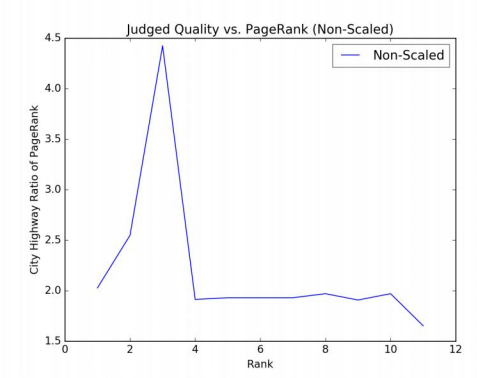

Figure 2: Non scaled results of the average city-to-highway ratio of PageRank on the different quality metromaps.

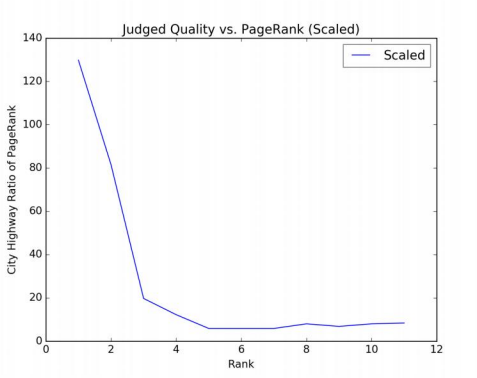

Figure 3: Scaled results of the average city-to-highway ratio of PageRank on the different quality metromaps.

This metric of the average scaled city-to-highway could help to optimize future metro-maps algorithms. This experiment may speak to a deeper property of the metro-maps. There may be a property of the size/density of information islands (clusters) that humans find easiest to understand. This may have uses outside of metro-maps such as creations of summaries from documents.

Term Intersection

An important general concept to metro-maps is that the nodes in the final graph, known as clusters, are related by concept. For example, the ideas of "Cuba" and "Florida" would likely come together in a cluster about "Miami". In regards to searching for jobs it may important to know how skills relate to a company. The individual articles that make up these clusters have commonalities with most of the terms that describe the cluster. When this concept is extended to jobs, skills, and companies, clusters that are described by key words of a job (such as skills or technologies) and also contain the name of a company, the nodes (in this case Wikipedia articles) in that cluster are likely to have commonalities of the skills and the company. Using the Wikipedia subset this intersection of terms was examined.

Starting with the Wikipedia subset starting with Java programing as the root node the intersection was examine for several different terms and companies. The intersection used is the set of all nodes that exist in clusters where all the terms are present. Several examples of skills/technologies intersected with companies are shown in table 4. The first entry interestingly has has pages such as processing and WebGL. These are not in HTML or CSS, but are for visualization. Nashorn is a JavaScipt engine, which JavaScipt is related to HTML and CSS in web development. Devoxx is interesting in that it is a conference on Java, Android, and HTML5. Amazon is not listed on the Wikipedia page, but a web search reveals that Amazon is involved in the conference and some of the software development kits for the Kindle Fire refer to Devoxx. The next entry for "Probabilty, Inference, and Google" has interesting results as well that do point more towards the mathematical side of development. DataNucleus is a data mangement tool in Java that is utilized in the persistence layer of Google's App Engine for Java. Also OpenMP, Gluegen (Java to low level application interface to bind Java and OpenGL), and OpenHMPP are related to cluster computing, which applications tend to be heavy in probability and/or inference. The final entry for "Java, Statistics, and Amazon" has interesting properties as well. Linux and GitHub are likely used in almost every organization that would use

Java. LAME, Bzip2, Gzip, and FAAC are all different compression methods (media specific, or general). This is interesting as they are not written in Java, but Java applications can make use of them. Additionally, Amazon has streaming services that likely make use of these technologies. Finally, Indoona is a cross platform instant messaging service with software development kits in Java where the Amazon Kindle has an Android application package for development. The relation was not clear from just the Wikipedia articles.

Terms	**Pages**
HTML, CSS, Amazon	Processing_(programming_language), WebGL, Nashorn_(JavaScript_engine), Devoxx
Probability, Inference, Google	DataNucleus, Scalatra, OpenMP, GlueGen, OpenHMPP, Java_openGL
Java, Statistics, Amazon	LAME, Bzip2, GitHub, Indoona, FAAC, Gzip, Linux

Table 4: Cluster intersection interesting terms.

The intersection of terms may help to discover connected information that was not known before. This could be helpful to find unknown convergent areas in disparate fields. Further development of metro-maps algorithm could be specifically targeted to find common points. For example, the academic journal network may elucidate new naturally occurring antibiotics by search for the intersection bacteria, resistance, and fungi.

Document Sequence Importance

The metro-maps algorithm maximizes coverage, coherence, and connectivity of the map. We conducted this experiment to test if metro-maps is able to do this as effectively if the chronology of the documents is unknown or mislabeled. We feel that getting favorable results in this experiment would mean we might be able to apply metro-maps across heterogeneous document sources e.g e-books, training video transcripts and Wikipedia, where the "sequence" of documents is not known. We conducted the experiment as follows:

1. We executed metro-maps algorithm on the Wikipedia subset with the white-list, a concept in the existing code to make sure terms stay on the coherent lines, consisting of all terms appearing in Job descriptions for Netflix.

2. We shuffled the document time-slices. We used this as our null model to generate maps and compare with the sequence biased model in 1.

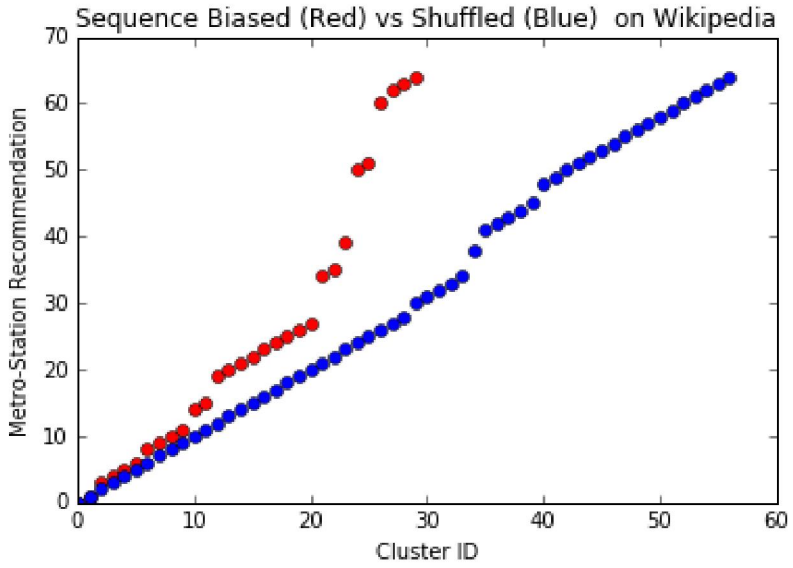

Figure 4: Comparison of Metro-Map best paths on Wikipedia data-set when sequence is known (Sequence Biased) vs. data-set when sequence is unknown (Unbiased).

In figure 4 the higher-slope of the red line means that this metro-line is able to navigate through more relevant subjects while maintaining maximal coherence, coverage, and connectivity. Hence the Sequence-Biased metro-maps

results in better coherent chains and provides a more efficient path for the learner to navigate through the map. We conclude that sequence of documents as a prior is an important factor for metro-maps in coherence maximization. Hence when using Metromaps for job specific course-work generation, it is important to encode temporal information into the document metadata for getting the most efficient path.

Use Case: Method to Increase Accuracy of Metromaps on Unlabeled Documents

Wikipedia, using a breadth first expansion, can provide a pseudo-temporal relationship between concepts in the form of links between pages. Hence one way of generating a time-line for unlabeled documents is to cluster the unlabeled documents with related Wikipedia pages. Once the time-sequence is known, we know from the above graph that metro-maps will perform much better against this data-set vs. an unlabeled dataset.

Node and Edge Importance for Netflix Job Descriptions

We used the metro-maps algorithm to evaluate how node and edge importance measures would vary when we used combined whitelists instead of a single whitelist. Initially we used a whitelist for a job description, "Senior Software Engineer, InfraEntertainment, and Media Industry." We then combined the whitelists of several Netflix job descriptions: "Senior Data Analyst Content Delivery Analyst", "Senior Security Intelligence Response Team", and "Senior Software Engineer Infrastructure" with the previous whitelist and then tested the response of metro-maps to the combined whitelist. We found that the metro-maps generated exactly the same map due to the similarity in keyword co-occurrences between the single and the combined whitelists for the identified clusters.

We used the following measures for identifying the most important node and edge with respect to betweenness centrality for edge and node, degree centrality and PageRank.

Centrality Measures and PageRank of the Combined Whitelist Map

PageRank	Degree Centrality	Degree
0.0549	0.1667	6

Table 5: Metrics for the most important node in the metro-map

For Node	For Edge
374.166	602.0

Table 6: Highest Betweenness Centrality Values

The Metro-maps algorithm for the combined whitelist found 4 lines. The lines found by the metro-maps algorithm were software engineering skills (keywords: create, define, requirements), platform programming skills (keywords: Java, Internet, Microsoft, Python), security and infrastructure (keywords: security, cloud), and a line, which showed a mix of the keywords in the three lines, but with a database focus (keywords: SQL, data, Hive). The most important node was found in the center of line 1.

The keywords that the most important node had were as expected indicating the set of common software skills that were applicable across Netflix jobs. Programming skills like Python, Android, Java, and generic software engineering skills like software, platform, infrastructure, and security also appeared in the most central cluster. Interestingly, the cluster that had a highest PageRank was also the cluster that had the highest betweenness centrality. Since we had an undirected graph, we could infer that this cluster was central to all the paths created by the whitelist against the Wikipedia pages, thereby contributing both to highest Pagerank scores as well as contributing the maximum to betweenness values. The second most central node which also formed the edge with the most central node for the maximum betweenness value showed more generic verbs like create, define, develop and also common software engineering skills like SQL, data, Microsoft, and web. Documents that formed the most central cluster were JBoss Seam, JBuilder, Jboss. These are tools mainly for application development IDEs and integration tools and also object-oriented programming languages like Julia and multi-paradigm C-based languages like Vala, which indicates the spectrum of common skills required for the Netflix jobs.

Hence, clusters that stand at the intersection of diverse skillset groups form the intersecting nodes and central edges and exhibit the highest centrality and PageRank values in the metro-map. Also, there is a possibility of specific

skillset groups to merge to form a cluster of broader skillsets for a given job description, and these clusters also show higher values for PageRank and centrality.

Conclusion

In conclusion, we have shown that the metro-maps algorithm is a versatile way to show human readable information in a compact form. A modified version of PageRank has the potential to be used for a metric to objectively score the produced maps, which may be helpful in integrating machine learning optimization techniques into the metro-maps algorithm. The intersection of terms can be used to help for jobs searches and for discovery of new related information show that metro-maps has both new commercial and academic potential. We showed that the performance of metro-maps marginally degrades if the document sequence is not known when calculating coherence. We proposed the use of Wikipedia as a reference to generate timelines for unlabeled data by mapping the unlabeled data to the breadth first step that matches the content of data the closest.

References

[1] Dafna Shahaf and Carlos Guestrin. Connecting the dots between news articles. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–632. ACM, 2010.

[2] Dafna Shahaf, Carlos Guestrin, and Eric Horvitz. Metro maps of science. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1122–1130. ACM, 2012.

[3] Dafna Shahaf, Carlos Guestrin, and Eric Horvitz. Trains of thought: Generating information maps. In *Proceedings of the 21st international conference on World Wide Web*, pages 899–908. ACM, 2012.

[4] Dafna Shahaf, Jaewon Yang, Caroline Suen, Jeff Jacobs, Heidi Wang, and Jure Leskovec. Information cartography: creating zoomable, large-scale maps of information. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1097–1105. ACM, 2013.

[5] Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596. ACM, 2013.