arXiv:1002.3195v1 [cs.AI] 17 Feb 2010

Efficiently Discovering Hammock Paths from Induced Similarity Networks

M. Shahriar Hossain, Michael Narayan, Naren Ramakrishnan
Department of Computer Science, Virginia Tech, Blacksburg, VA 24061
Email: {msh, mnarayan, naren}@vt.edu

Abstract

Similarity networks are important abstractions in many information management applications such as recommender systems, corpora analysis, and medical informatics. For instance, in a recommender system, by inducing similarity networks between movies rated similarly by users, we can aim to find the global structure of connectivities underlying the data, and use the network to posit connections between given entities. We present an algorithmic framework to efficiently find paths in an induced similarity network *without materializing the network in its entirety*. Our framework introduces the notion of 'hammock' paths which are generalizations of traditional paths in bipartite graphs. Given starting and ending objects of interest, it explores candidate objects for path following, and heuristics to admissibly estimate the potential for paths to lead to a desired destination. We present three diverse applications, modeled after the Netflix dataset, a broad subset of the PubMed corpus, and a database of clinical trials. Experimental results demonstrate the potential of our approach for unstructured knowledge discovery in similarity networks.

1. Introduction

There is significant current interest in modeling and understanding network structures, especially in online social communities, web graphs, and biological networks. We focus here on (unipartite) similarity networks induced from bipartite graphs, such as a movies $\times$ people dataset. Two movies can be connected if they have been rated similarly by sufficient number of people; this is the basis for the popular item-based recommendation algorithms [4]. A similarity network thus exposes an indirect level of clustering, community formation, and organization that is not immediately apparent from the bipartite network.

Similarity networks are key abstractions in recommender systems but they also find uses in other information management applications such as collection analysis and medical informatics. We focus on how these networks can be used for *exploratory discovery*, i.e., to see how similarities can be composed to reach potentially distant entities. For instance, how is the movie 'Roman Holiday' (a romantic classic) connected to 'Terminator 3' (a Sci-Fi movie)? What are the in-between movies and waypoints that help link these two disparate movies? Are these waypoints characteristic to the domain or to the dataset? For recommender systems, these questions are especially germane to tasks such as serendipitous recommendation, gift buying, and characterizing the movie watching interests of its users.

In addition to recommender systems, we consider similarity networks induced from literature and semi-structured sources of data. Here path finding has applications to literature-based discovery (similar to the ARROWSMITH program [12]) and clinical diagnosis. For instance, how is congestive heart failure related to kidney disease? The discovered waypoints correspond to possibilities by which different disease response pathways interact with each other.

Admittedly, these questions can be answered by first inducing the similarity network and running shortest path queries over it, but we desire to find paths *without materializing the entire network*. This would be wasteful because only a subnetwork is likely to be traversed in response to a query. Instead, we seek to

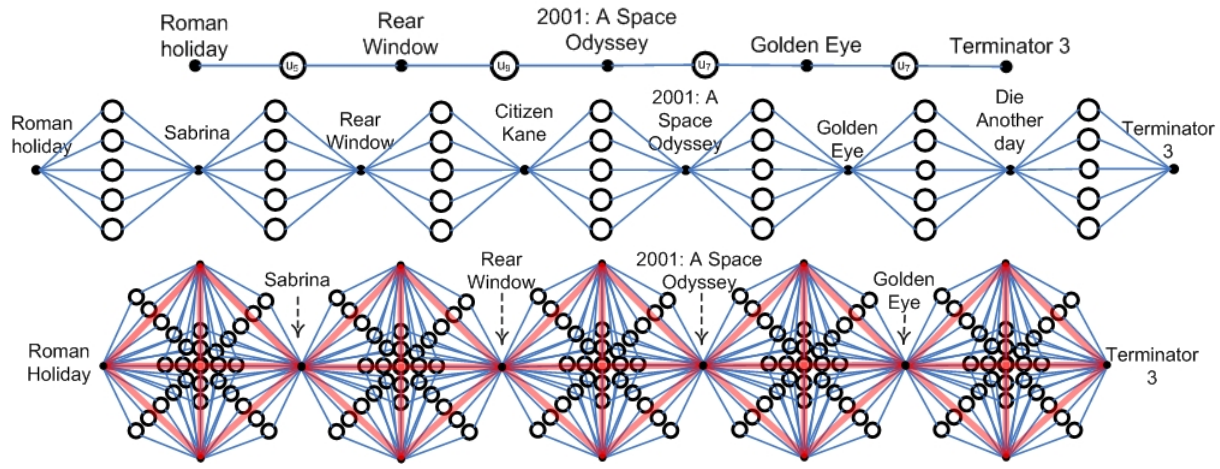


Figure 1: (*Top*) A simple path (hammock width=1) beginning at romantic classic "Roman Holiday" and ending at the Sci-Fi & Fantasy movie "Terminator III". (*Middle*) A hammock path between the same entities but with hammock width=5. (*Bottom*) A hammock path between the same entities with an additional clique size requirement of 4. Observe that each hammock inside the cliques continues to have width 5. Notice that the path length increases from (*Top*) to (*Middle*) and then drops in (*Bottom*).

pre-process the original bipartite graph into suitable data structures that can then be harnessed to find chains of connections.

In this paper, we present an algorithmic framework for traversing paths using the notion of 'hammock' paths which are generalization of traditional paths. Our framework is exploratory in nature so that, given starting and ending objects of interest, it explores candidate objects for path following, and heuristics to admissibly estimate the potential for paths to lead to a desired destination. Our key contributions are:

1. We formulate the path finding problem over similarity networks in terms of hammocks and cliques, two intuitive-to-understand constructs for navigating similarity networks. Hammocks are ways to impose tighter requirements over individual links (leading to longer paths) and cliques are ways to coalesce multiple links (resulting in shorter paths).
2. We present an algorithmic framework that finds hammock paths in both binary datasets (bipartite graphs) and vector-valued datasets (weighted bipartite graphs). Our framework uses a concept lattice representation as an in-memory data structure to organize the search for paths and introduces admissible heuristics that quickly prune out unpromising paths.
3. We present compelling experimental results on three diverse datasets: the Netflix dataset of movie ratings, a significant portion of the PubMed corpus, and a database of clinical trials. Experimental results demonstrate the scalability and accuracy of our approach as well as its potential for unstructured knowledge discovery.

2. Background

The formulations we study can be intuitively understood using Fig. 1. A simple path between movies can be induced through co-raters, as shown in Fig. 1 (top): 'Roman Holiday' was rated by user u_5 who also rated 'Rear Window' which was also rated by u_9, and so on. In practice, paths are induced between two movies only if a sufficient number of people have rated them and rated them similarly. This leads to a sequence of 'hammocks' as shown in Fig. 1 (middle). A final level of generalization is to organize the hammocks into groups so that we traverse cliques of movies (with a hammock between every pair of them), see Fig. 1 (bottom). By finding such hammocks and traversing them systematically, similarities can

'diffuse' to reach possibly distant entities. Note that the path length increases as the hammock requirement is strengthened and then decreases as the clique requirement is imposed. Work closest to ours can be found in the graph modeling, redescription mining, and lattice-based information retrieval literature.

Graph modeling: The use of the word 'hammock' for induced similarity networks appears to have been first made in [8] although this work does not aim to find paths. The notion of kNC-plots was introduced in [6] where k denotes what we refer to as hammock width in this paper. Rather than finding local paths, this paper is focused on finding the global connectivity structure of the induced networks as the hammock width is increased. It is also restricted to binary spaces whereas we focus on vector valued spaces as well. Random intersection graphs [13] are a class of theoretical models proposed in the random graph community. These models randomly assign each vertex with a subset of a given set and posit edges if these subsets intersect. Under these modeling assumptions, connectivity and other properties of these graphs can be characterized [2].

Redescription mining: Redescriptions, a pattern class introduced in [11], induce subsets of data that share strong overlap. A hammock in our notation can be viewed as a redescription between the objects it connects. Kumar et al. [5] study the problem of 'storytelling' over a space of descriptors, which is the goal of finding a sequence (story) of redescriptions between two subsets by positing intermediate subsets that share sufficient overlap with their neighbors in the story. Although this is similar to our objective, the heuristic innovations in [5] are restricted to binary spaces as well and cannot find paths of the same expressiveness (hammocks organized into cliques) and with the same efficiencies as done here.

Lattice based information retrieval: The use of concept lattices as organizing data structures for fast retrieval, query, and data mining is not new [9, 10]. Our work is different in both the theoretical framework by which the lattice concepts are used to build chains and in their uses/applications.

3. Problem Formulation

We begin by formally defining the space over which similarity networks are induced.

Definition 1. A dataset $\mathcal{D} = (\mathcal{O}, \mathcal{F}, \mathcal{V})$ is a set of objects $\mathcal{O}$, a set of features $\mathcal{F}$, and a relation $\mathcal{V} \subseteq \mathcal{O} \times \mathcal{F}$.

We can thus think of each $f \in \mathcal{F}$ as inducing a subset of $\mathcal{O}$, which we call $o(f)$. We further require that the objects induced by $\mathcal{F}$ form a covering of $\mathcal{O}$, i.e. $\bigcup_{f \in \mathcal{F}} o(f) = \mathcal{O}$. In the applications studied here, the $(\mathcal{O}, \mathcal{F})$ are (movies, users), (documents, terms), and (clinical trials, keywords), respectively, with the obvious meaning of $\mathcal{V}$ in each case. In Fig. 2 we introduce a dataset which we will use as a running example. Here the objects are movies ($\mathcal{O} = \{m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}$) and the features are users ($\mathcal{F} = \{u_A, u_B, u_C, u_D, u_E, u_F, u_G\}$).

A note on notation: We use calligraphic $\mathcal{O}$ and $\mathcal{F}$ to represent the complete object and feature sets for some database. Lower case letters o and f are used to represent individual members of $\mathcal{O}$ and $\mathcal{F}$, respectively. To represent subsets of $\mathcal{O}$ and $\mathcal{F}$ we use upper case letters O and F, resp.

For a feature set $F \subseteq \mathcal{F}$, we overload notation and define the operator $o(F) : \mathcal{F} \to \mathcal{O}$ as the set of objects which are associated with all of the features in F, i.e.,

$$o(F) = \bigcap_{f \in F} o(f)$$

We also define a parallel operator $f(O)$ as the set of features associated with the object set O. In Fig. 2, we have $o(\{u_A, u_D, u_E\}) = \{m_1, m_3, m_5, m_7\}$ and $f(\{m_1, m_3, m_5, m_7\}) = \{u_A, u_D, u_E\}$.

We can now define the closure operator $c : 2^{\mathcal{F}} \to 2^{\mathcal{F}}$ as $c(F) = f(o(F))$. A feature set F is closed if and only if $c(F) = F$. (A parallel closure operator exists on $2^{\mathcal{O}}$, but is not necessary for our purposes.) Using our running example from Figure 2 we see that feature set $\{u_A, u_D, u_E\}$ is closed, whereas $\{u_A, u_D\}$ is not, since $c(\{u_A, u_D\}) = \{u_A, u_D, u_E\} \neq \{u_A, u_D\}$.

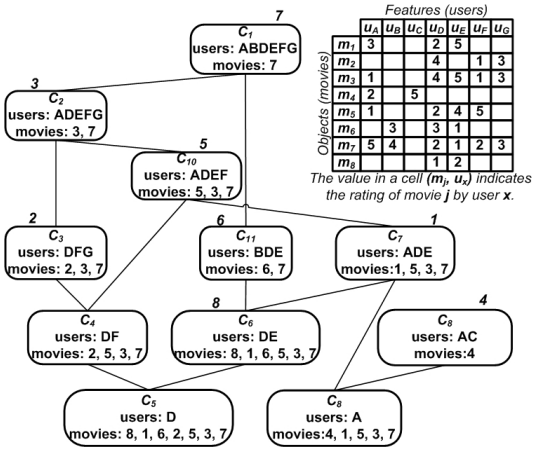


Figure 2: A dataset and its concept lattice.

Definition 2. For each feature $f \in \mathcal{F}$, we define a predicate $p_f(o)$ such that $p_f(o)$ is true if and only if $o \in o(f)$. The set of all such predicates is denoted by $\mathcal{P}$. A *descriptor* is a boolean expression over $\mathcal{P}$.

A descriptor thus provides a mechanism to specify a set of objects which correspond to some Boolean expression on features. Again referring to Figure 2 we might define a descriptor $u_A(o) \vee u_B(o)$, which would correspond to all the movies which were rated by user A, by user B, or by both; it induces the set $\{m_1, m_3, m_5, m_6, m_7\}$. It is easy to see that we can create a descriptor for the objects in $o(F)$ by creating a conjunction over the features in F. For simplicity, we often speak of descriptor F for some feature set, which refers to this conjunction.

We employ the notion of *redescriptions* introduced by Ramakrishnan et al. [11] to capture similarities between descriptors. If for descriptors F and F', where F is tautologically distinct from F' (i.e. $F \vee \neg F'$ is not a tautology), it is the case that $o(F) = o(F')$, we say that F and F' are *redescriptions* of each other, as they induce the same set of objects. In other words, a redescription provides (at least) two different ways to describe a given object set. We next introduce the concept of a redescription set:

Definition 3. A *redescription set* for a descriptor F, $R_F = (O', F')$ is a tuple where $O' = o(F)$ and $F' = c(F)$.

Fig. 2 shows many redescription sets organized alongside a concept lattice (to be defined soon). One redescription set is: $\mathbf{C}_7 = (\{m_1, m_3, m_5, m_7\}, \{u_A, u_D, u_E\})$, which would correspond to both descriptors $\{u_A, u_D, u_E\}$ and $\{u_A, u_D\}$.

We say that a descriptor F is a relaxation of descriptor F', denoted by $(F \leq F')$, if the feature set $F = \{f_1, f_2, ..., f_m\} \subseteq \{f'_1, f'_2, ..., f'_n\} = F'$. In Fig. 2, $\{u_A, u_B\} \leq \{u_A, u_B, u_C\}$. The following is easy to verify:

Lemma 1. $F \leq F' \rightarrow o(F) \supseteq o(F')$.

Finally, we bring in the notion of a *concept lattice*, the lattice defined over the redescription sets using the operator $\leq$ with a join of $\mathcal{F}$ and a meet of $\emptyset$. Returning once again to our running example, Figure 2 shows a concept lattice with the lower nodes being relaxations of the higher nodes in the lattice. In this space we say that a descriptor F is a child of F' if and only if $F \leq F'$ and $\forall F'' : F \leq F'' \leq F' \rightarrow (F'' = F) \vee (F'' = F')$. Thus in our running example $\mathbf{C}_{10}$ has two children: $\mathbf{C}_4$ and $\mathbf{C}_7$.

Definition 4. The *Jaccard coefficient* between two objects is defined as

$$J(o_1, o_2) = \frac{|f(o_1) \cap f(o_2)|}{|f(o_1) \cup f(o_2)|}$$

The Jaccard coefficient is thus a measure of how similar two objects are based upon how many features they share in proportion to their overall size.

Definition 5. The *Soergel distance* between two objects o_1 and o_2 is given by

$$D(o_1, o_2) = \frac{|f(o_1)| + |f(o_2)| - 2|f(o_1) \cap f(o_2)|}{|f(o_1)| + |f(o_2)| - |f(o_1) \cap f(o_2)|}$$

Unlike the Jaccard coefficient (which is a similarity measure), the Soergel distance is a true distance measure: it is exactly 0.0 when the objects o_1 and o_2 have exactly the same features, is symmetric, and obeys the triangle inequality. We also note that the Soergel distance is exactly 1.0 when the induced feature sets are disjoint.

We now define two types of paths—clique paths and hammock paths—which we use to constrain our path generation.

Definition 6. A *hammock* of width w is a tuple $(o_1, o_2) \in \mathcal{O} \times \mathcal{O}$ where $|f(o_1) \cap f(o_2)| = w$.

A hammock is thus a pair of objects which share at least w common features. Another way to think of a hammock is as a redescription set containing two objects ($\{o_1\}$ and $\{o_2\}$), with a feature set which contains at least w features. Observe that a hammock is a basic unit of similarity modeling in recommender systems and many other applications: it posits similarity in one domain using relationships with another domain. We now define hammock paths composed of hammocks.

Definition 7. A *hammock path* with width w, from object o_1 to object o_t, denoted by $H^w(o_1, o_t)$, is a series of objects $\langle o_1, o_2, ..., o_{t-1}, o_t \rangle$ such that $\forall i : 1 \leq i \leq t-1$ the pair (o_i, o_{i+1}) is a hammock of width at least w.

We now define a *clique* which extends the concept of a hammock to a set of objects rather than a single pair:

Definition 8. A k-clique, $q^{k,w}$, is a set of k objects $O = \{o_1, o_2, \ldots, o_k\}$ such that $\forall o_i, o_j \in O : |f(o_i) \cap f(o_j)| \geq w$, for some width w. The function Υ maps a clique to the set of k objects contained in said clique.

Note that a k-clique is defined over a collection of $k(k-1)/2$ hammocks of width at least w; the overlap threshold applied to every pair of objects of a clique ensures that there is at least as high an overlap between any two objects in the clique.

Definition 9. A k-clique path $Q^{k,w}(o_1, o_t)$ is a series of $t-1$ consecutive k-cliques $Q = \langle q_1, q_2, \ldots, q_{t-1} \rangle$ such that $o_1 \in \Upsilon(q_1)$, $o_t \in \Upsilon(q_{t-1})$, and $\forall q_i \in Q : \Upsilon(q_i) \cap \Upsilon(q_{i+1}) \neq \emptyset$.

The problem we seek to address can now be given as a pair of constraints upon the clique and hammock paths between two objects, formally defined as follows:

Problem 1. *Given $\mathcal{D} = (\mathcal{O}, \mathcal{F})$, start and end objects $o_1 \in \mathcal{O}$ and $o_t \in \mathcal{O}$, we seek to find a chain/series of objects $S_{1,t} = \langle o_1, o_2, \ldots, o_t \rangle$ where $S_{1,t}$ is a hammock path $H^w(o_1, o_t)$ for some width w, and further, there is some k-clique path $Q^{k,w}(o_1, o_t) = \langle q_1, q_2, \ldots, q_{t-1} \rangle$ where $\forall i : 1 \leq i \leq t-1$, $o_i \in \Upsilon(q_i)$.*

4 Extension to Vector Spaces

To accommodate vector spaces, we generalize Definition 1 so that a database $\mathcal{D} = (\mathcal{O}, \mathcal{F}, V)$ is now a tuple defined over a set of objects $\mathcal{O}$, features $\mathcal{F}$, and a function $V : \mathcal{O} \times \mathcal{F} \rightarrow \mathbb{R}$. This helps us go beyond simple binary associations to record the strength of the association (e.g., rating values, term weights) or other auxiliary continuous-valued data. (Our running example of Figure 2 contains movie ratings by users.) The *weighted Soergel distance* between two objects o_1 and o_2 is defined by

$$\mathbb{D}(o_1, o_2) = \frac{\sum_{f \in \mathcal{F}} |V(o_1, f) - V(o_2, f)|}{\sum_{f \in \mathcal{F}} \max(V(o_1, f), V(o_2, f))}$$

which reduces to the unweighted case if $V(o, f) = 1$ for any object $o \in o(f)$ and $V(o, f) = 0$ for any object $o \notin o(f)$. Definitions 6, 7, 8, 9 get similarly generalized. In place of the hammock width constraint w, we use a weighted Soergel distance threshold θ that must be satisfied between the vectors that aim to form hammocks, hammock paths, cliques, or clique paths. In place of the notation $H^w(o_1, o_t)$, we use $H^\theta(o_1, o_t)$, and so on. The new problem becomes:

Problem 2. *Given $\mathcal{D} = (\mathcal{O}, \mathcal{F}, V)$, start and end objects $o_1 \in \mathcal{O}$ and $o_t \in \mathcal{O}$, we seek to find a chain/series of objects $S_{1,t} = \langle o_1, o_2, \ldots, o_t \rangle$ where $S_{1,t}$ is a hammock path $H^\theta(o_1, o_t)$ for some distance θ, and further, there is some k-clique path $Q^{k,\theta}(o_1, o_t) = \langle q_1, q_2, \ldots, q_{t-1} \rangle$ where $\forall i : 1 \leq i \leq t-1$, $o_i \in \Upsilon(q_i)$.*

5. Algorithms

Our overall methodology is based on using the concept lattice to structure the search for paths. Recall that the two parameters influencing the quality of the path—hammock width and clique size—impose a duality. The hammock width is posed over feature sets but the clique size is over objects. We use the clique size to prune the concept lattice during construction (by incorporating it as a support constraint) and the hammock width to select candidates for dynamic construction of paths. There are three key computational stages: (i) construction of the concept lattice, (ii) generating promising candidates for path following, and (iii) evaluating candidates for potential to lead to destination. Of these, the first stage can be viewed as a startup cost that can be amortized over multiple path finding tasks.

We adopt the CHARM-L [14] algorithm of Zaki for constructing concept lattices and mining redescriptions. The second and third stages are organized as part of an A* search algorithm that begins with the starting object, uses the concept lattice to identify candidates satisfying the hammock and clique size requirements, and evaluates them heuristically for their promise in leading to the end object. In practice, we will place a limit on the branching factor (b) of the search, thus sacrificing completeness for efficiency. We showcase these steps in detail below, including the construction of admissible heuristics.

5.1. Successor Generation

Successor generation is the task of, given an object, using the hammock and clique size requirements, to identify a set of possible successors for path following. Note that this does not use the end object in its computation. We study three techniques for successor generation:

1. Cover Tree Nearest Neighbor,
2. Nearest Neighbors Approximation (NNA), and
3. k-Clique Near Neighbor (kCNN).

The first technique is targeted toward finding paths when the clique size requirement is 2 (top and middle paths of Fig. 1). That is, this technique is able to generate b singleton successors, where b is the branching factor of the search. The second two techniques concentrate on paths of any clique size, such as the bottom path of Fig. 1. Instead of generating singleton successors, the NNA and kCNN algorithms are able to generate successor-sets, where each of these sets constitutes a candidate k-clique with the given object.

5.1.1 Cover Tree Nearest Neighbor

The cover tree [1] is a data structure for fast nearest neighbor operations in a space of objects organized alongside any distance metric (here, we use the Soergel distance [7]). The space complexity is $O(\|\mathcal{O}\|)$, i.e., linear in the object size of the database. A nearest neighbor query requires logarithmic time in the object space $O\left(c^{12}\log\left(n\right)\right)$ where c is the expansion constant associated with the featureset dimension of the dataset (see [1] for details).

5.1.2 Nearest Neighbors Approximation (NNA)

The second mechanism we use for successor generation is to approximate the nearest neighbors of an object using the concept lattice. We use the Jaccard coefficient between two objects as an indicator to inversely (and approximately) track the Soergel distance between the objects. In order to efficiently calculate an object's nearest neighbors, however, we cannot simply calculate the Jaccard coefficient between it and every other object. We harness the concept lattice to avoid wasteful comparisons. We first define a predicate τ on redescriptions and objects.

Algorithm 1 NNA(o)

Input: An object $o \in \mathcal{O}$
  $fringe \leftarrow \top(o)$ order by feature set size
  $prospects \leftarrow \emptyset$
  while $fringe \neq \emptyset$ **do**
    $r \leftarrow$ dequeue from *fringe*
    while $prospects \neq \emptyset$ **do**
      $o' \leftarrow$ head *prospects*
      if $J(o, o') > \frac{|\text{Items}(o)|}{|\text{Items}(r)|}$ **then**
        yield o'
        dequeue *prospects*
      else
        break
      end if
    end while
    for all $r' \in$ ChildrenOf r **do**
      add r' to *fringe* order by feature set size
      for all o' in $o(r')$ **do**
        add o' to *prospects* order by $J(o, o')$
      end for
    end for
  end while

Definition 10. For redescription $R_F = (O, F)$ and object o, $\tau(o, R_F)$ if and only if $o \in o(F)$ and there is no F' such that $F \leq F'$ and $o \in o(F')$.

Informally, we can say $\tau(o, R_F)$ if F is on the edge of redescriptions which contain the object o. In Figure 2, using object m_1 the only feature set F for which $\tau(m_1, R_F)$ would evaluate to true is $F = \{u_A, u_D, u_E\}$. Note that if no support threshold (clique constraint) is given, there will always be exactly one such feature set for each object. When using a support threshold greater than one, this is no longer true; so we now define the set $\top(o, 2^{\mathcal{F}})$ for object o.

Definition 11. For object o and all redescriptions $2^{\mathcal{F}}$ in a concept lattice containing o, $\top(o, 2^{\mathcal{F}}) = \{F : F \in 2^{\mathcal{F}} \wedge \tau(o, R_F)\}$.

We generally omit the $2^{\mathcal{F}}$ where it is clear from context. The set $\top(o)$ is then the set of all redescriptions which form the upper edge in the concept lattice where o appears. All the objects in Figure 2 have singleton sets for $\top$, however, if we change the support threshold from one object to five objects, then the object m_1 will have $\top(m_1) = \{\{u_A\}, \{u_D, u_E\}\}$.

A formal description of our NNA algorithm is shown in Algorithm 1.

Definition 12. $\mathrm{NNA}_k(o)$ is the k^{th} object returned by the NNA algorithm when run on input object o.

Theorem 1. *Given a database with object set $\mathcal{O}$ with size $|\mathcal{O}| = n$ containing object $o \in \mathcal{O}$, $\forall i, j \in \mathbb{Z}$: $1 \leq i \leq j \leq n \rightarrow J(o, \mathrm{NNA}_i(o)) \geq J(o, \mathrm{NNA}_j(o))$.*

(Proof omitted due to space constraints.) In other words, NNA returns better approximate redescriptions of an object first. This is still, however, an approximation since it uses the Jacccard coefficient rather than the Soergel distance.

5.1.3 k-Clique Near Neighbor (kCNN)

The basic idea of the kCNN approach is, in addition to finding a good set of successor nodes for a given object o, to be able to have sufficient number of them so that, combinatorially, they contribute a desired number of cliques. With a clique size constraint of k, it is not sufficient to merely pick the top k neighbors of the given object, since the successor generation function expects multiple clique candidates. (Note that, even if we picked the top k neighbors, we will still need to subject them to a check to verify that every pair satisfies the hammock width constraint.) Given that this function expects b clique candidates, minimum number m of candidate objects to identify can be cast as the solution to the inequalities:

$$\binom{m-1}{k} < b \text{ and } \binom{m}{k} \geq b$$

The object list of each concept of the lattice is ordered in the number of features (e.g., see Fig. 2) and this aids in picking the top m candidate objects for the given object o. We pick up these m candidate objects for o from the object list of the concept containing the longest feature set and redescription set containing o. Note that, in practice, the object list of each concept is much larger than m and as a result kCNN does not need to traverse the lattice to obtain promising candidates. kCNN thus forms combinations of size k from these m objects to obtain a total of b k-cliques. Since m is calculated using the two inequalities, the total number of such combinations is equal to or slightly greater than b (but never less than b). Each clique is given an average distance score calculated from the distances of the objects of the clique and the current object o. This aids kCNN in returning a priority queue of exactly b candidate k-cliques.

5.2. Evaluating Candidates

We now have a basket of candidates that are close to the current object and must determine which of these has potential to lead to the destination. We present two operational modes to rank candidates: (i) the normal mode and (ii) the mixed mode.

5.2.1 Normal Mode

The normal mode is suitable for the general case where we have all the objects and features resident in the database. The primary criteria of optimality for the A* search procedure is the cumulative Soergel distance

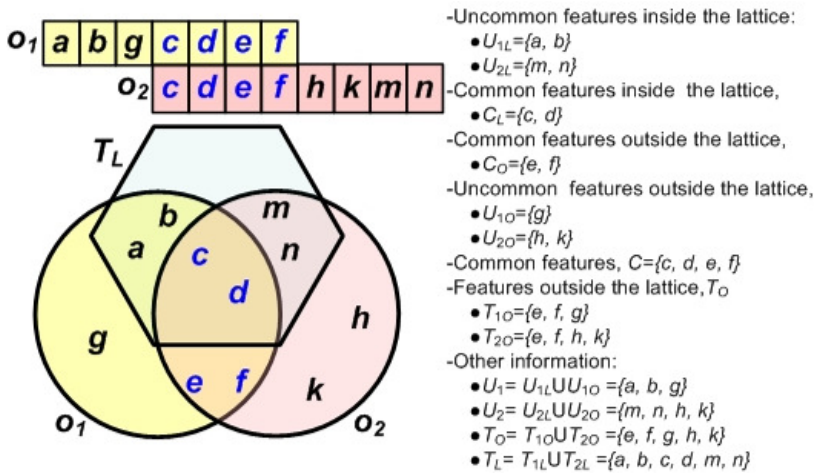


Figure 3: Distribution of common and uncommon features of objects o_1 and o_2 inside and outside the concept lattice. (The hexagon indicates the concept lattice).

of the path. We use the straight line Soergel distance for the heuristic. It is easy to see that this can never oveestimate the cost of a path from any object o to the goal (and is hence admissible), because the Soergel distance maintains the triangle inequality.

5.2.2 Mixed Mode

The mixed mode distance measure is effective for large datasets where only important information are stored but other information are removed from the system after recording some of their aggregated information, to save space and cost of establishment. With the mixed mode approach, for simplicity, we assume that all the information about items outside the concept lattice are absent but some of their aggregated information like number of features truncated are provided. Fig. 3 shows the distribution of common and uncommon features of objects o_1 and o_2 inside and outside a concept lattice. The mixed mode Soergel distance is given by:

$$D_{\text{mixed}}(o_1, o_2) = \frac{|U_{1L}| + |U_{2L}|}{|U_{1L}| + |U_{2L}| + |C_L| + |T_O|}$$

where the terms are defined in Fig. 3.

Theorem 2. *$D_{mixed}(o_1, o_2)$ never overestimates the original Soergel distance $D(o_1, o_2)$.*

Proof: Omitted due to more general Theorem 3 later.

5.2.3 Mixed Mode in Vector Spaces

However, the mixed mode becomes more complex in the vector space model than its set model. Figure 4 shows our formula for the mixed mode in the vector space. Similar to the unweighted one, we assume that all the information about features of objects outside the concept lattice are absent in the weighted mixed

$$\mathbb{D}_{\text{mixed}}(o_1, o_2) = \frac{|U_{1L}|\times\text{minw}(o_1)+|U_{2L}|\times\text{minw}(o_2)+\sum_{f\in C_L}|V(o_1,f)-V(o_2,f)|}{|U_{1L}|\times\text{minw}(o_1)+|U_{2L}|\times\text{minw}(o_2)+\sum_{f\in C_L}\max(V(o_1,f),V(o_2,f))+|T_O|\times\max(\text{maxw}(o_1),\text{maxw}(o_2))}$$

Figure 4: The mixed mode Soergel distance equation.

mode approach. But some of their aggregated information like minimum (minw) and maximum (maxw) weights are provided.

Consider the set of features T_O that do not appear in the lattice due to the support threshold of the concept lattice, *minsup*. Some features of T_O can be common to both objects o_1 and o_2. $|U_{1O}|$ and $|U_{2O}|$ are the numbers of uncommon features in objects o_1 and o_2, which are thus outside the frequent concept lattice. Length $|T_O|$ is a known variable due to the recorded aggregated information, but $|U_{1O}|$ and $|U_{2O}|$ are unknown. This is why $|U_{1O}|$ and $|U_{2O}|$ do not appear in $\mathbb{D}_{\text{mixed}}(o_1, o_2)$. For $\mathbb{D}_{\text{mixed}}(o_1, o_2)$, we consider that all the features of T_O (i.e., features outside the lattice) are common in both objects o_1 and o_2 and all these features have the same weight which is $\max(\text{maxw}(o_1), \text{maxw}(o_2))$.

Theorem 3. *$\mathbb{D}_{mixed}(o_1, o_2)$ never overestimates the original Soergel distance $\mathbb{D}(o_1, o_2)$.*

Proof: Let the numerator of $\mathbb{D}(o_1, o_2)$ and $\mathbb{D}_{\text{mixed}}(o_1, o_2)$ be $\eta^{1,2}$ and $\eta^{1,2}_{\text{mixed}}$ respectively. Similarly, let $\varpi^{1,2}$ and $\varpi^{1,2}_{\text{mixed}}$ be the denominators of $\mathbb{D}(o_1, o_2)$ and $\mathbb{D}_{\text{mixed}}(o_1, o_2)$. It is clear that $\eta^{1,2}_{\text{mixed}} \leq \eta^{1,2}$. Therefore, it suffices to show that $\varpi^{1,2}_{\text{mixed}} \geq \varpi^{1,2}$.

For ease of derivation, we define the following notation :

$$
\begin{aligned}
\alpha &= |U_{1L}| \times \text{minw}\,(o_1) + |U_{2L}| \times \text{minw}\,(o_2) \\
\beta &= \sum_{f \in C_L} \max(V(o_1, f), V(o_2, f)) \\
\zeta &= \sum_{f \in (T_O - U_{1O} - U_{2O})} \max(V(o_1, f), V(o_2, f)) \\
\chi &= |U_1| \times \text{minw}\,(o_1) + |U_2| \times \text{minw}\,(o_2) \\
\xi &= |U_{1O}| \times \text{minw}\,(o_1) + |U_{2O}| \times \text{minw}\,(o_2) \\
\varrho &= |T_O| \times \max\left(\text{maxw}\,(o_1), \text{maxw}\,(o_2)\right)
\end{aligned}
$$

Therefore we have, $\varpi^{1,2} = \chi + \beta + \zeta$. Now, $\varpi^{1,2}_{\text{mixed}} = \alpha + \beta + \varrho = \chi + \beta + \zeta - \xi + \varrho - \zeta = \varpi^{1,2} + \varrho - \zeta - \xi = \varpi^{1,2} + L$, where $L = \varrho - \zeta - \xi$. Since $L \geq 0$ is always true, $\varpi^{1,2}_{\text{mixed}} \geq \varpi^{1,2}$. Therefore, $\mathbb{D}_{\text{mixed}}(o_1, o_2) \leq \mathbb{D}(o_1, o_2)$. □

6. Experimental Results

We present our experimental results on four datasets (see Table 1) using a 64-bit Windows Vista Intel Core2 Quad CPU Q9450 @ 2.66GHz and 8 GB physical memory.

6.1. Evaluating successor generation strategies

The goal of our first experiment was to assess the number of nodes explored by the A* search and the time taken as a function of the discovered path length, as a function of different successor generation

Table 1: Dataset characteristics.

Dataset	# obj.	# feat.	# relations	Sparsity
Synthetic	1,000	3,679	430,477	99.88
Netflix	17,770	480,189	100,480,507	99.98
PubMed	161,693	133,252	2,226,616	99.99
Cl. Trials	66,526	75,656	907,759	99.99

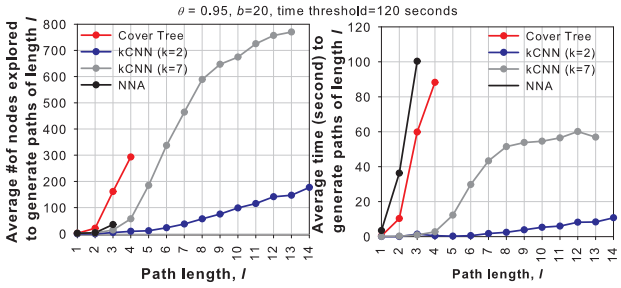


Figure 5: Synthetic dataset: Comparison between successor generation strategies.

strategies. For this purpose, we defined a synthetic dataset involving 1000 randomly selected movies from the Netflix dataset and aimed to generate more than 50,000 similarity paths between randomly selected pairs, with a θ threshold of 0.95 and a branching factor bound (b) of 20. We introduced a time threshold of 120 seconds in the A* search and Fig. 5 depicts the results of only successful searches. Around 80% of these searches failed when using the the cover tree and NNA approaches, but the kCNN approach was able to either successfully generate paths or to declare that no path exists in less than 120 seconds. As Fig. 5 shows, the cover tree and NNA terminate early due to the applied time constraint but kCNN generated long paths of length 14 and 13 with k=2 and 7, respectively. The runtime trends shown in Fig. 5 (right) also mirror the number of nodes explored in Fig. 5 (left).

This result is not surprising, as the cover tree algorithm does not factor in the clique constraint, thus preventing it from taking advantage of the search space reduction that this constraint provides. NNA does take advantage of this constraint, however, and it generates a strict ordering on the Jaccard's coefficient over the cliques, whereas kCNN simply generates some b candidate cliques. In practice, the kCNN relaxation results in the discovery of candidate cliques more rapidly than the NNA algorithm, while still remaining accurate as in both cases a post processing step is necessary to determine if a given candidate does in fact meet the search threshold. Through the rest of this paper, we thus use the kCNN algorithm as it generally provides the best performance of the three algorithms.

6.2. Netflix dataset

Viewing movies as objects and users as features, we construct the concept lattice for the Netflix dataset with a support constraint of 20%. The resulting lattice contains 5,884 concepts, 120 of whom were leaves and the rest had child concepts. We picked 50,000 pairs of movies and attempted to generate hammock paths between them.

Figure 6 shows our experimental results with varying clique size and fixed distance threshold. From the

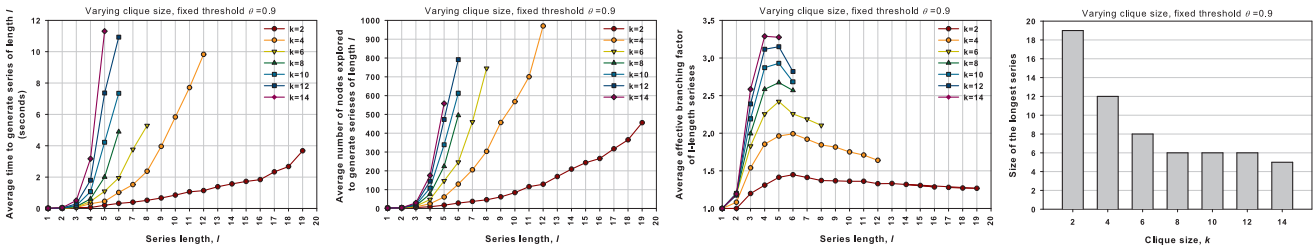
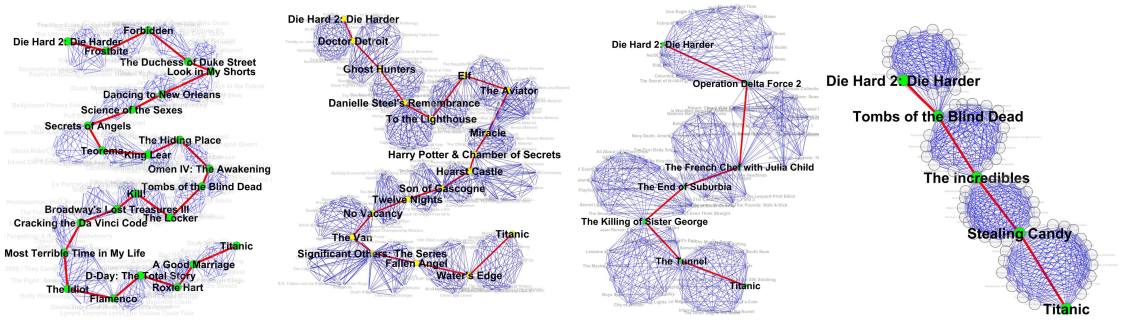


Figure 6: Netflix: Explorations by varying clique size requirement at a fixed Soergel threshold (θ=0.9). The first three plots show average run time, number of nodes explored, and effective branching factor, as a function of path lengths, for different clique size requirements. The fourth plot shows that the size of the longest path reduces as the clique size increases.



Clique size=9 Clique size=12 Clique size=15 Clique size=18

Figure 7: Netflix: Hammock paths from 1943 classic drama "Titanic" to the action movie "Die Hard 2: Die Harder" for four different clique size requirements.

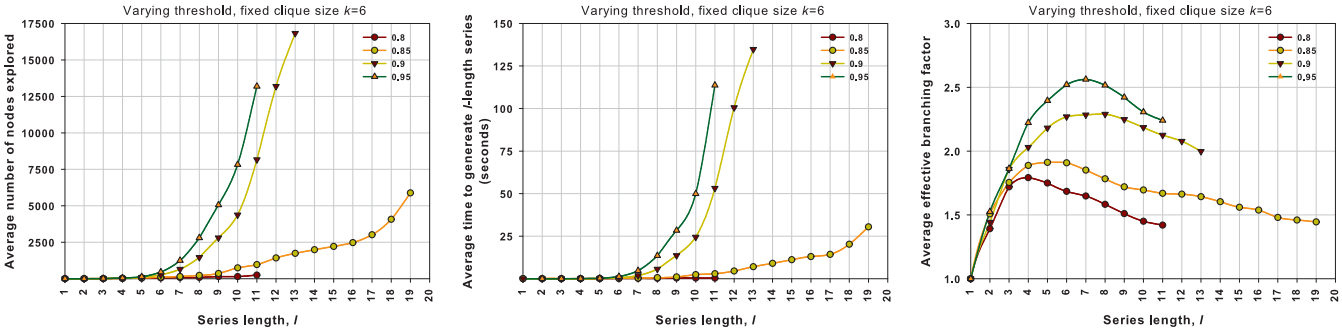
graph at the left, it is evident that the performance of our kCNN algorithm depends on the clique size since the graph shows that hammock paths with lower clique sizes are generated faster than those with higher clique sizes. The tendency of the run time basically follows the required number of nodes explored to generate the paths (the second graph). The third graph shows the corresponding trends of effective branching factor. It is evident that the higher the clique size the worse (or larger) the effective branching factor is. Effective branching factor is a measure to understand the size of the traversed search space compared with corresponding breadth first search (BFS). Therefore it is a measure of the efficacy of the heuristic in pruning out unwanted paths. Although the effective branching factor becomes worse with larger clique size, it generates smaller hammock paths. Thus, the lengths of the paths are also affected by the clique size requirement. The plot at the right depicts the lengths of the hammock paths as a function of clique size. It demonstrates that our algorithm has the capability to generate longer chains with smaller cliques. For example, Figure 7 gives hammock paths between the 1943 classic drama "Titanic" and the action movie "Die Hard 2: Die Harder", for four different clique sizes (k=9, 12, 15 and 18).

6.3. PubMed Abstracts

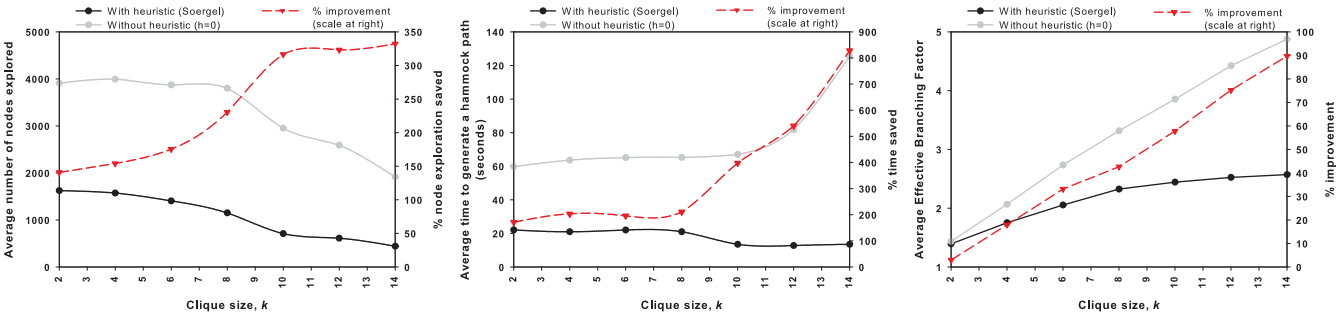
In our PubMed case study, we view paper abstracts as objects and terms as features. We curated 161,693 publications related to several cytokines, transcription factors and feedback molecules and modeled only their titles and abstracts. There were around 133,252 unique stemmed terms in the dataset after the removal of stop words, numerals, DNA sequences, and special characters. Again, for randomly selected pairs of papers, we discovered over a million hammock paths.

Figure 8(a) shows the trends of number of nodes explored, time to generate paths, and effective branching factor, as a function of path length. Although Figure 8(a) shows that the average number of nodes explored is large with higher threshold, it is not necessary that this behavior would remain the same for other datasets. The result can in fact be opposite in some datasets where number of paths generated are not lessened by reducing θ, especially when each object has a large number of features.

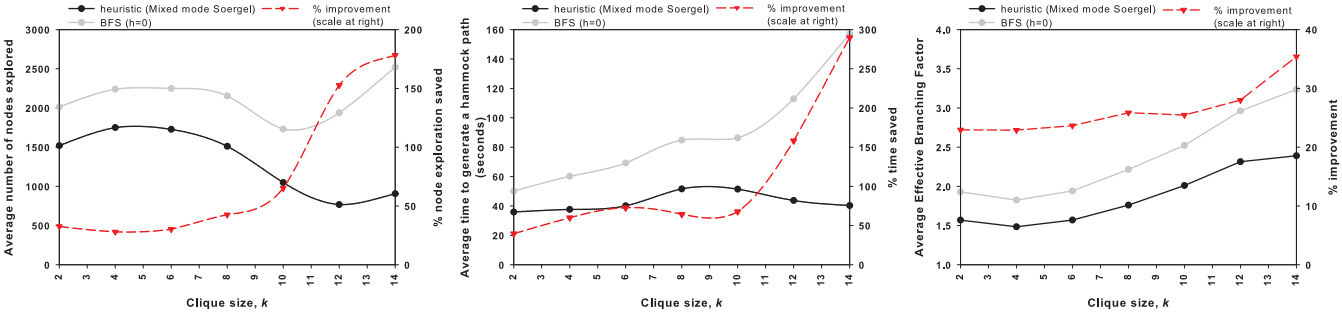
Figure 8(b) (left) shows that the use of Soergel distance heuristic saves significant object exploration over the vanilla BFS. The larger the clique size, the higher the percentage by which the heuristic reduces the number of node explorations. It shows that the use of the straight line Soergel distance as the heuristic saves more than 300% node exploration by the A* procedure over the BFS (h=0), for a clique size k=14. The saved amount is more than 100% even with the smallest clique size k=2. The average time to generate hammock path also reduces due to the saved node exploration. Figure 8(b) (middle) shows that the heuristic saves more than 800% runtime with clique size k=14. Even with a clique size k=2, the savings are near 200%. In the best case with k=14, the heuristic also improves the effective branching factor by 90% over



(a) Varying threshold θ, fixed clique size (k=6). (Left) Number of nodes explored by the A* procedure. It shows that the application of lower thresholds results in a lower number of explored nodes, as a function of path length. (Middle) The higher the distance threshold, the higher the run time to generate paths. (Right) Lower distance threshold has the trend to generate paths with low effective branching factor.



(b) The use of Soergel distance heuristic results in better performance in terms of number of nodes explored, runtime and effective branching factor. The dashed line in each of the plots shows the percentage improvement due to the use of heuristic over plain BFS with h=0.



(c) Mixed mode operation: (Left) the mixed mode heuristic provides lesser traversal in the similarity network over the corresponding BSF. (Middle) the mixed mode heuristic saves larger percentage of time with larger cliques over the corresponding BFS. (Right) The effective branching factor improvement is higher with larger cliques.

Figure 8: Some illustrative experimental results using the PubMed dataset.

the BFS shown in the plot at the right. In the worst case with k=2, it offers around 4% improvement of the effective branching factor.

Despite the use of truncated dataset, Figure 8(c) shows that the mixed mode gains due to the heuristic over the BFS have a similar trend to the normal mode of Figure 8(b). Therefore, the mixed mode offers a practical mechanism to provide the best possible gains from lossy datasets without time consuming remodeling of the vector space (e.g., [5] uses costly remodelling as a post processing step).

Table 2: Clinical trials: A hammock path connecting congestive heart failure and kidney complications.

Trial ID	Short Title of the Trial
00103519	Study of DITPA in Patients With Congestive Heart Failure.
00696631	European Trial of Dronedarone in Moderate to Severe Congestive Heart Failure.
00697086	Study of Dronedarone in Atrial Fibrillation.
00744874	Ablation of the Pulmonary Veins for Paroxysmal Afib.
00807586	Corticosteroid Pulse After Ablation.
00030563	Surgery With or Without Radio frequency Ablation Followed by Irinotecan in Treating Patients With Colorectal Cancer that is Metastatic to the Liver.
00003753	Floxuridine, Dexamethasone, and Irinotecan After Surgery in Treating Patients With Liver Metastases From Colorectal Cancer.
00005818	SU5416 and Irinotecan in Treating Patients With Advanced Colorectal Cancer.
00002828	Chemotherapy With Raltitrexed and Fluorouracil in Treating Patients With Advanced Colorectal Cancer.
00449137	Arsenic Trioxide, Fluorouracil, and Leucovorin in Treating Patients With Stage IV Colorectal Cancer That Has Relapsed or Not Responded to Treatment.
00124605	Arsenic Trioxide and Pamidronate in Treating Patients With Advanced Solid Tumors or Multiple Myeloma.
00302627	Pamidronate, Vitamin D, and Calcium for the Bone Disease of Kidney and Heart Transplantation.
00074516	Kidney Transplantation in Patients With Cystinosis.

6.4. Clinical Trials

We curated more than 60 thousand clinical trials from clinicaltrials.gov and concentrated on the purpose and description of each trial. Here trials are objects and terms are features. The concept lattice we used had around a thousand unique concepts and was generated with minsup=10%. Due to space restrictions, we show qualitative rather than quantitative results here.

Table 2 describes a significant chain of trials connecting two disparate clinical studies related to congestive heart failure and kidney transplantation in patients with cystinosis. (One accepted practice to assess the statistical significance of discovered chains is, for each step in the path, to assess the likelihood of overlap for the given descriptor sizes using the hypergeometric distribution and attribute a p-value after FDR corrections such the Benjamini-Hochberg procedure.) The chain starts with heart failure trials and goes through studies on atrial fibrillation (abnormal heart rhythm), cardiac ablation, colorectal cancer, advanced solid tumors, and eventually reaches clinical study on kidney transplantation in patients with cystinosis. Such connections between cardio-vascular disease and kidney failures are an intense topic of current research (e.g., see [3]).

7. Discussion

We have presented an efficient algorithmic approach to discover hammock paths in similarity networks without inducing these networks in their entirety. The experimental results have demonstrated scalability, effectiveness of our heuristics, and ability to yield domain-specific insight. We posit that our approach can be a useful information exploration tool for understanding the structure of connectivities underlying boolean and vector-valued association datasets. Future work is geared toward incorporating additional distance measures and defining new compressed representations of datasets that can serve multiple uses, from concept modeling to distance estimation.

References

[1] A. Beygelzimer, S. Kakade, and J. Langford. Cover Trees for Nearest Neighbor. In *ICML '06*, pages 97–104, 2006.

[2] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification—A Technique for Speeding up Dynamic Graph Algorithms. *J. ACM*, 44(5):669–696, 1997.

[3] J. Gill et al. Outcomes of Simultaneous Heart-Kidney Transplant in the US: A Retrospective Analysis Using OPTN/UNOS Data. *Am J Transplant.*, 9(4):844–852, 2009.

[4] G. Karypis. Evaluation of Item-Based Top-N Recommendation Algorithms. In *CIKM '01*, pages 247–254, 2001.

[5] D. Kumar, N. Ramakrishnan, R. F. Helm, and M. Potts. Algorithms for Storytelling. In *KDD '06*, pages 604–610, 2006.

[6] R. Kumar, A. Tomkins, and E. Vee. Connectivity Structure of Bipartite Graphs via the KNC-plot. In *WSDM '08*, pages 129–138, 2008.

[7] A. Leach and V. Gillet. *Introduction to Chemoinformatics*. Springer, 2007.

[8] B. J. Mirza, B. J. Keller, and N. Ramakrishnan. Studying Recommendation Algorithms by Graph Analysis. *J. Intell. Inf. Syst.*, 20(2):131–160, 2003.

[9] S. Narayanaswamy and R. Bhatnagar. A Lattice-Based Model for Recommender Systems. In *ICTAI '08*, pages 349–356, 2008.

[10] G. S. Pedersen. A Browser for Bibliographic Information Retrieval, based on an Application of Lattice Theory. In *SIGIR '93*, pages 270–279, 1993.

[11] N. Ramakrishnan, D. Kumar, B. Mishra, M. Potts, and R. F. Helm. Turning CARTwheels: an Alternating Algorithm for Mining Redescriptions. In *KDD '04*, pages 266–275, 2004.

[12] N. R. Smalheiser and D. R. Swanson. Using ARROWSMITH: A Computer-assisted Approach to Formulating and Assessing Scientific Hypotheses. *Computer Methods and Programs in Biomedicine*, 57(3):149–153, 1998.

[13] D. Stark. The vertex degree distribution of random intersection graphs. *Random Struct. Algorithms*, 24(3):249–258, 2004.

[14] M. J. Zaki and N. Ramakrishnan. Reasoning About Sets Using Redescription Mining. In *KDD '05*, pages 364–373, 2005.