Algorithmica (2007) 48: 363–374
DOI: 10.1007/s00453-007-0155-7

Algorithmica

Finding Pathway Structures in Protein Interaction Networks[1]

Songjian Lu,[2] Fenghui Zhang,[2] Jianer Chen,[2] and Sing-Hoi Sze[2,3]

Abstract. The increased availability of data describing biological interactions provides important clues on how complex chains of genes and proteins interact with each other. Most previous approaches either restrict their attention to analyzing simple substructures such as paths or trees in these graphs, or use heuristics that do not provide performance guarantees when general substructures are analyzed. We investigate a formulation to model pathway structures directly and give a probabilistic algorithm to find an optimal path structure in $O(4^k n^{2t} k^{t+\log(t+1)+2.92} t^2)$ time and $O(n^t k \log k + m)$ space, where n and m are respectively the number of vertices and the number of edges in the given network, k is the number of vertices in the path structure, and t is the maximum number of vertices (i.e., "width") at each level of the structure. Even for the case $t = 1$ which corresponds to finding simple paths of length k, our time complexity $4^k n^{O(1)}$ is a significant improvement over previous probabilistic approaches. To allow for the analysis of multiple pathway structures, we further consider a variant of the algorithm that provides probabilistic guarantees for the top suboptimal path structures with a slight increase in time and space. We show that our algorithm can identify pathway structures with high sensitivity by applying it to protein interaction networks in the DIP database.

Key Words. Probabilistic algorithm, Pathway structure, Protein interaction network.

1. Introduction.

By representing a biological network as a graph in which vertices represent genes or proteins and edges represent interactions between them, many algorithms have been proposed to study substructures in these graphs in order to understand the organization of interacting components within these networks [1]–[11]. The increased availability of these networks that describe interactions at a genome scale presents serious computational challenges since they can be very large, with thousands of vertices and tens of thousands of edges. In protein interaction networks, one important biological problem is to find chains of proteins that belong to a functional pathway, which corresponds to finding simple paths in the network with closely interacting proteins and is already a very difficult *NP*-hard problem [12]. In reality, biological pathways are not linear since there may be multiple interacting paths within a pathway. In order to understand these complex interactions, a more accurate model is to find a collection of closely related chains of proteins that form a pathway structure, which can be achieved by identifying a subgraph of the given network that includes the chains.

To address this problem, Steffen et al. [4] employed an exhaustive search procedure to find short simple paths in a given network that contain functional related proteins and

[1] This work was supported in part by the National Science Foundation under Grants CCR-0311590 and CCF-0430683.
[2] Department of Computer Science, Texas A&M University, College Station, TX 77843, USA. {sjlu,fhzhang,chen,shsze}@cs.tamu.edu.
[3] Department of Biochemistry & Biophysics, Texas A&M University, College Station, TX 77843, USA.

Received March 1, 2006; revised November 9, 2006. Communicated by P. Berman, B. DasGupta, and J. Liang. Online publication July 13, 2007.

combine top scoring paths into a path structure. Kelley et al. [5] developed a probabilistic algorithm that allows the identification of longer optimal simple paths in $k!n^{O(1)}$ time, where n is the number of vertices and k is the path length. Scott et al. [11] proposed an improved probabilistic algorithm for finding simple paths that runs in $5.44^k n^{O(1)}$ time by using the color coding technique [13] and allowed the identification of more complicated substructures such as trees and series–parallel graphs. A few other approaches allow the identification of arbitrary biological substructures, but these algorithms either can only identify small substructures optimally or have to use heuristics to find large substructures. Koyutürk et al. [6] used a branch-and-bound procedure to identify small frequent subgraphs that occur in many networks. Hu et al. [7] employed a few graph-theoretic reductions to find dense subgraphs that occur in many networks. Sharan et al. [9] defined the notion of a network alignment graph and used a greedy heuristic to identify conserved subnetworks that occur in multiple species. Koyutürk et al. [10] defined the notion of local network alignment and used a greedy heuristic to identify conserved substructures in two graphs.

We are interested in developing algorithms with at least a probabilistically guaranteed performance that allows systematic investigation of pathway structures. A popular previous technique is to employ a two stage approach that first identifies high scoring simple paths and then combines these paths into a graph (that is a subgraph of the original network) to represent the path structure [4], [5], [11]. One drawback of this approach is that potential interactions between related paths are not taken into account. We investigate a different formulation that models a pathway structure directly as a leveled structure of proteins in which proteins from adjacent levels are connected together to represent a collection of closely related chains of proteins. We develop a divide-and-conquer algorithm to find an optimal path structure with high probability under a scoring scheme that models characteristics of biological pathways. Since the scoring scheme may not perfectly model biological reality, we also consider a variant of the algorithm that provides probabilistic guarantees for the top suboptimal path structures with a slight increase in time and space. We show that our algorithm can identify biological pathway structures by applying it to protein interaction networks in the DIP database [14].

2. Problem Formulation.

Let k be the number of vertices in the target path structure and let t be the maximum number of vertices at each level of the structure. We use the following notion of a (k, t)-path to represent a path structure.

DEFINITION 1. Given an undirected graph G, two disjoint sets of vertices S_1 and S_2 are said to be *connected* if each vertex in S_1 is adjacent to at least one vertex in S_2 and each vertex in S_2 is adjacent to at least one vertex in S_1. A (k, t)-*path* is a subgraph of G with k vertices that are partitioned into l disjoint subsets $S_1, S_2, \ldots, S_l$ such that (1) $|S_i| \leq t$ for $1 \leq i \leq l$, and (2) S_i is connected to S_{i+1} for $1 \leq i < l$. We denote the (k, t)-path as $[S_1, S_2, \ldots, S_l]$, and call S_1 and S_l the *ends* of the (k, t)-path (see Figure 1).

A (k, t)-path represents a path structure of size k with l levels in which each level is of size at most t, with the property that each vertex is included in a simple path of length l within the path structure. With this formulation, multiple interacting paths within a

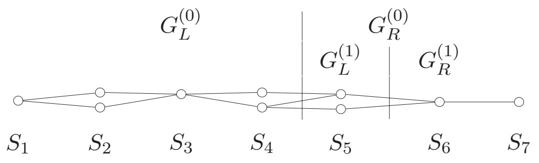

Fig. 1. Illustration of the definition of a (k, t)-path and the probabilistic divide-and-conquer algorithm. The parameters are $k = 10, t = 2$, and $l = 7$. S_1 and S_7 are the ends of the (k, t)-path. Only edges between adjacent levels are shown (there may be many other edges in G that connect vertices from non-adjacent levels). One iteration of step 3 of the algorithm in Figure 2 is shown in which G is randomly partitioned into two parts $G_L^{(0)}$ and $G_R^{(0)}$ that also subdivides the (k, t)-path into two smaller path structures of roughly equal sizes. The recursion within $G_L^{(0)}$ returns a set of path structures, in which $[S_1, S_2, S_3, S_4]$ is among one of them. $G_R^{(0)}$ is further partitioned into $G_L^{(1)}$ and $G_R^{(1)}$. The path structure $[S_1, S_2, S_3, S_4]$ is then concatenated to $[S_5]$ before $[S_6, S_7]$ is finally added.

pathway can be modeled and proteins within the same level in a path structure are likely to play similar roles within the pathway. By imposing appropriate vertex or edge weights in G, it becomes possible to investigate these interacting paths in biological networks directly. Although the formulation only considers edges between adjacent levels in the structure, no restrictions are imposed on edges that can exist between non-adjacent levels. Furthermore, complicated path structures can be represented even when t is small (e.g., $t \leq 2$ or 3). To study paths between two vertices in G, one can impose a source and a sink in the path structure (i.e., both the first and the last levels have only one vertex). Since vertices in different levels are distinct, a (k, t)-path then represents multiple simple paths of length l between the source and the sink. Note that in this formulation, the number of levels l is not a parameter and the best path length l (with respect to a given (k, t) pair) will be found automatically during the computation. Since it is sometimes necessary to find paths of a specific length l that form a path structure, an alternative strategy is to consider l as a parameter. We will show that a variant of our algorithm can be used to find multiple (k, t)-paths for each value of l simultaneously without a significant increase in time and space.

Our formulation is different from the series–parallel graph formulation in [11] which defines a different type of non-linear structure. Our representation of the path structure is also different from a path-decomposition of bounded pathwidth [15] which allows vertex groupings that may overlap and has additional constraints on edges within a vertex group. Since very extensive overlaps can occur within a vertex group in a path-decomposition, much more complicated structures than multiple simple paths are allowed, which may not be biologically useful since meaningless paths with very high scores that contain many repeated vertices can be included.

Even for the case $t = 1$, the problem is difficult since it corresponds to the *NP*-hard problem of finding simple paths of length k [12]. Despite significant efforts, there is no practical deterministic approach that guarantees that the optimal path is found unless the path length is very short [4]. The best previous deterministic algorithm uses the color coding technique [13], which, when coupled with the dynamic programming technique in [11], gives an algorithm that runs in $d^k n^{O(1)}$ time, where d is an impractically large constant. The best previous probabilistic algorithm combines the probabilistic version of the color coding technique in [13] with the dynamic programming technique in [11] to give an approach that guarantees to find the optimal path with high probability and runs in $(2e)^k n^{O(1)} = 5.44^k n^{O(1)}$ time and $O(nk2^k + m)$ space, which are both exponential in k. Kelley et al. [5] gave the best previous probabilistic algorithm that uses polynomial space by imposing acyclic edge orientations and runs in $k!n^{O(1)}$ time and $O(m+n)$ space. For the special case $t = 1$, our probabilistic algorithm that runs in $4^k n^{O(1)}$ time and $O(nk \log k + m)$ space offers a significant improvement over these probabilistic approaches, with a much lower constant in the exponential part of the time complexity while requiring only polynomial space.

3. Probabilistic Algorithm. For simplicity, we first consider finding an arbitrary (k, t)-path in G. Suppose that such a (k, t)-path exists and t is small. The idea is to use a divide-and-conquer approach to partition G randomly into two subgraphs. Consider this partition a success if it subdivides a (hidden) (k, t)-path into two smaller path structures of roughly equal sizes, which results in two subproblems of finding (k', t)-paths with k' roughly equal to $k/2$ that can connect together to form a (k, t)-path. Continue to partition recursively into smaller subgraphs until the base case $k \le t$ is reached, in which the (k, t)-paths can be enumerated exhaustively as follows.

LEMMA 2. *Let G be an undirected graph with n vertices and let $k \le t$. There are at most $(2n)^k$ (k, t)-paths in G and these (k, t)-paths can be constructed in $O(k(2n)^k) = O(t(2n)^t)$ time.*

PROOF. Let $S = \{v_1, \ldots, v_k\}$ be any set of k vertices in G. We first consider how many (k, t)-paths can be obtained from these vertices in S. Let π be a permutation of S. We insert "separators" between the vertices in π to partition π into non-empty segments (we require that at most one separator be inserted between two vertices in π). We call this a *partitioned permutation* of S. Any (k, t)-path formed from S can be represented by a partitioned permutation of S, in which segments correspond to levels in the (k, t)-path. Although each (k, t)-path formed from S may be represented by more than one partitioned permutation of S, each partitioned permutation of S can represent at most one valid (k, t)-path. Thus the number of partitioned permutations of S gives an upper bound on the number of (k, t)-paths that can be obtained from S. Fix a permutation π and let $T(k)$ be the total number of partitioned permutations that can be formed from π. Since for each i, $0 < i \le k$, there are $T(k - i)$ partitioned permutations in which the first segment consists of the first i vertices in π, we have the following recurrence:

$$T(k) = T(k-1) + T(k-2) + \cdots + T(0),$$

which gives $T(k) \le 2^k$ (since $T(0) = 1$). Since there are $k!$ permutations of the vertices in S, there are at most $2^k k!$ partitioned permutations of the vertices of S. Thus at most $2^k k!$ (k, t)-paths can be obtained from S. Since there are $\binom{n}{k} \le n^k/k!$ sets of k vertices in G, we conclude that the total number of (k, t)-paths in G is bounded by $(n^k/k!)(2^k k!) = (2n)^k$. These (k, t)-paths can be easily constructed in $O(k(2n)^k) = O(t(2n)^t)$ time by the exhaustive enumeration method described above. $\square$

```
Algorithm find-paths($G$, $P_0$, $k$, $t$)
{Assume that no path structure in $P_0$ contains a vertex in $G$}

1.    $P = \emptyset$;
      if $k \leq t$ then
2.        for each $(S, S', k, t)$-path $p$ in $G$ do
              if $P_0 = \emptyset$ then
2.1.              add $p$ to $P$ if no $(S', k, t)$-path is in $P$;
              else
2.2.              for each $(S'', k', t)$-path $p'$ in $P_0$ do
                      if $S''$ is connected to $S$ then
2.3.                      concatenate $p$ and $p'$ into an $(S', k + k', t)$-path $p''$;
2.4.                      add $p''$ to $P$ if no $(S', k + k', t)$-path is in $P$;
      else
3.        loop $2.51 \cdot 2^k$ times do
3.1.          randomly partition the vertices in $G$ into two parts $V_L$ and $V_R$;
3.2.          let $G_L$ and $G_R$ be the subgraphs induced by $V_L$ and $V_R$ respectively;
3.3.          for $i = k - \lfloor (k + t)/2 \rfloor$ to $\lfloor (k + t)/2 \rfloor$ do
3.4.              $P_L$ = find-paths($G_L$, $P_0$, $i$, $t$);
                  if $P_L \neq \emptyset$ then
3.5.                  $P_R$ = find-paths($G_R$, $P_L$, $k - i$, $t$);
3.6.                  for each $(S, k + k', t)$-path $p$ in $P_R$ do
3.7.                      add $p$ to $P$ if no $(S, k + k', t)$-path is in $P$;
4.    return $P$;
```

Fig. 2. Illustration of the find-paths algorithm.

To allow the merging of two disjoint smaller path structures p and p' to form a larger path structure, one end of p must be connected to an end of p'. We introduce the following notations. Let S and S' be two sets of vertices of size at most t in G. An (S, k, t)-path is a (k, t)-path in which one end is S. Similarly, an (S, S', k, t)-path is a (k, t)-path in which one end is S and the other end is S'. Our algorithm recursively constructs a set P_0 of (k', t)-paths for a subgraph G_0, where for each set S of size at most t in G_0, at most one (S, k', t)-path is stored in P_0. Suppose that we have constructed such a set P_0 of (k', t)-paths for a subgraph G_0 and let G be a graph that shares no vertices with G_0. The algorithm find-paths(G, P_0, k, t) returns a set P of $(k + k', t)$-paths in the induced subgraph of G formed by the vertices in G_0 and G, with at most one $(S, k + k', t)$-path in P for each vertex set S of size at most t in G, and each $(S, k + k', t)$-path is a concatenation of a (k', t)-path in P_0 and an (S, k, t)-path in G. The initial call to the algorithm is find-paths(G, $\emptyset$, k, t), which returns a set of (k, t)-paths in G. Figure 2 illustrates the details of the algorithm, where we assume that no path structure in P_0 contains a vertex in G (see also Figure 1). Step 2 of the algorithm represents the base case in which exhaustive enumeration is possible and step 3 represents the recursive partition step.

THEOREM 3. *Let G be an undirected graph with n vertices and m edges and let S be a set of vertices of size at most t in G. If G contains an (S, k, t)-path, then with probability larger than $1 - 1/e > 0.632$, the set P returned by find-paths$(G, \emptyset, k, t)$ contains* *an (S, k, t)-path. The algorithm find-paths$(G, \emptyset, k, t)$ runs in $O(4^k n^{2t} k^{t+\log(t+1)+2.92} t^2)$ time and $O(n^t k \log k + m)$ space.*

PROOF. We prove the following claims by induction on k.

1. If $P_0 = \emptyset$ and G has an (S, k, t)-path, then with probability larger than $1 - 1/e$, the set P returned by find-paths(G, P_0, k, t) contains an (S, k, t)-path.
2. If P_0 is a non-empty set of (k', t)-paths, and G has an (S, k, t)-path such that its other end is connected to a (k', t)-path in P_0, then with probability larger than $1 - 1/e$, the set P returned by find-paths(G, P_0, k, t) contains an $(S, k + k', t)$-path that is a concatenation of an (S, k, t)-path in G and a (k', t)-path in P_0.

The claims are obviously true when $k \le t$ since all (k, t)-paths in G are exhaustively enumerated. We let $k > t$ and first consider the case when $P_0 = \emptyset$. Suppose that $[S_1, S_2, \dots, S_{k_1}, S_{k_1+1}, \dots, S_l]$ is an (S, k, t)-path in G, where each S_i is a level in the (S, k, t)-path of size at most t, $S_l = S$, $\sum_{i=1}^{k_1} |S_i| = d_1 \le (k + t)/2$ and $\sum_{i=k_1+1}^{l} |S_i| = d_2 \le (k + t)/2$. Such a choice of k_1 is always possible since $|S_i| \le t$ for all i. With probability $1/2^k$, step 3.1 puts the vertices in $S_1, S_2, \dots, S_{k_1}$ into V_L and the vertices in $S_{k_1+1}, \dots, S_l$ into V_R. In this case, G_L contains the (S_{k_1}, d_1, t)-path $[S_1, \dots, S_{k_1}]$, and G_R contains the $(S, k - d_1, t)$-path $[S_{k_1+1}, \dots, S_l]$. By the inductive hypothesis, with probability larger than $1 - 1/e$, P_L from step 3.4 contains an (S_{k_1}, d_1, t)-path. When this is the case, the $(S, k - d_1, t)$-path $[S_{k_1+1}, \dots, S_l]$ in G_R has its other end S_{k_1+1} connected to the (S_{k_1}, d_1, t)-path in P_L. By the inductive hypothesis, with probability larger than $1 - 1/e$, P_R from step 3.5 contains an (S, k, t)-path. Thus the probability ρ that an (S, k, t)-path is added to P in step 3 is larger than

$$\frac{(1 - 1/e)^2}{2^k} > \frac{0.632^2}{2^k} > \frac{1}{2.51 \cdot 2^k}.$$

When $P_0 \ne \emptyset$, we follow the same argument as before except that we require that the (S, k, t)-path in G has its other end connected to a (k', t)-path in P_0, P_L contains an $(S_{k_1}, d_1 + k', t)$-path that is a concatenation of an (S_{k_1}, d_1, t)-path in G_L and a (k', t)-path in P_0, and P_R contains an $(S, k + k', t)$-path that is a concatenation of an $(S, k - d_1, t)$-path in G_R and a $(d_1 + k', t)$-path in P_L.

Since step 3 loops $2.51 \cdot 2^k$ times, the overall probability that the set P returned by find-paths(G, P_0, k, t) contains an (S, k, t)-path when P_0 is empty or an $(S, k + k', t)$-path when P_0 is not empty is

$$1 - (1 - \rho)^{2.51 \cdot 2^k} > 1 - \left(1 - \frac{1}{2.51 \cdot 2^k}\right)^{2.51 \cdot 2^k} > 1 - \frac{1}{e}.$$

This completes the first part of the proof.

We now analyze the complexity of the algorithm. Let $T(k)$ be the time complexity of find-paths(G, P_0, k, t). When $k \le t$, by Lemma 2, there are at most $(2n)^t$ (k, t)-paths in G, and these (k, t)-paths can be constructed in $O(t(2n)^t)$ time. Since P_0 contains at most one (S, k', t)-path for each set S of size at most t, the number of path structures in P_0 is at most $\binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{t} \le n^t$. For each (S'', k', t)-path in P_0 and each (S, S', k, t)-path in G, since both S'' and S are of size at most t, it takes $O(t^2)$ time to check if S'' and S are connected. Thus we have $T(k) = O(t(2n)^t + (2n)^t n^t t^2) = O(n^{2t}2^t t^2)$ when $k \leq t$.

When $k > t$, since there are at most n^t path structures in P_R, steps 3.6–3.7 take $O(kn^t)$ time. We have the following recurrence:

$$\begin{aligned} T(k) &= 2.51 \cdot 2^k \sum_{i=k-\lfloor (k+t)/2 \rfloor}^{\lfloor (k+t)/2 \rfloor} (T(i) + T(k-i) + ckn^t) \\ &\leq 2.51(t+1)2^k(2T(\lfloor (k+t)/2 \rfloor) + ckn^t), \end{aligned}$$

where c is a constant. We can assume that $ckn^t \leq T(\lfloor (k+t)/2 \rfloor)$. Thus

$$T(k) \leq 7.53(t+1)2^k T(\lfloor (k+t)/2 \rfloor) \leq 7.53(t+1)2^k T(k/2 + t/2).$$

To solve this, note that a general form derived from the recurrence is

$$T(k) \leq (7.53(t+1))^i 2^{g(k,t,i)} T(k/2^i + t/2^i + t/2^{i-1} + \cdots + t/2),$$

where

$$\begin{aligned} g(k,t,i) &= k + (k/2 + t/2) + (k/2^2 + t/2^2 + t/2) + \cdots \\ &\quad + (k/2^{i-1} + t/2^{i-1} + t/2^{i-2} + \cdots + t/2). \end{aligned}$$

Let $i = \log k$ and note that $T(k) = O(n^{2t}2^t t^2)$ when $k \leq t$, we have

$$T(k) = O((7.53(t+1))^{\log k} 2^{2k+t(\log k - 1)} n^{2t} 2^t t^2) = O(4^k n^{2t} k^{t+\log(t+1)+2.92} t^2).$$

We now analyze the space complexity of the algorithm. Since each recursive call of find-paths uses $O(n^t k)$ space (mainly for the sets P_L, P_R, and P) and the depth of recursion is $\log k$, find-paths$(G, \emptyset, k, t)$ uses $O(n^t k \log k + m)$ space. □

The algorithm is thus practical when k is moderately large and t is small, with 4^k being the dominating term in the time complexity. Note that as t becomes larger relative to k, the running time increases to compensate for the more unbalanced subdivisions of the target path structure into two smaller path structures of different sizes in step 3.3. To achieve an arbitrarily small error bound ε, we can run the algorithm r times so that r satisfies $1/e^r < \varepsilon$ (e.g., for $\varepsilon = 0.001$, we have $r = 7$). To allow (k,t)-paths for each value of l to be found independently, instead of storing at most one (S,k,t)-path in steps 2.4 and 3.7, at most one (S,k,t)-path is stored for each value of l. This increases the time and space complexity by at most a factor of k.

4. Optimization for $t \leq 2$. Since protein interaction graphs are often sparse, we replace steps 2 and 2.2–2.4 ($k \leq t$ and $P_0 \neq \emptyset$) of our algorithm by the following procedure to lower the running time when $t = 1$: concatenate each $(\{v\}, k', t)$-path in P_0 (if it exists) and each edge (v, w) in G to obtain a $(\{w\}, k'+1, t)$-path, while storing at most one $(\{w\}, k'+1, t)$-path for each w. When $t = 2$, a slightly more complicated procedure is used: for each pair of edges (v_1, w_1) and (v_2, w_2) in G such that v_1 and v_2 appear among the path structures in P_0 and w_1 and w_2 are in G (v_1 and v_2 and w_1 and w_2 are not necessarily distinct), concatenate the $(\{v_1, v_2\}, k', t)$-path in P_0 (if it exists) with $\{w_1, w_2\}$ to obtain a $(\{w_1\}, k'+1, t)$-path if $w_1 = w_2$ or a $(\{w_1, w_2\}, k'+2, t)$-path if $w_1 \neq w_2$, while storing at most one path structure for each $\{w_1, w_2\}$. This procedure gives $(S, k'+1, t)$-paths for all S of size 1 and $(S, k'+2, t)$-paths for all S of size 2, with at most one path structure for each S. Since $k \leq t = 2$, the other $(S, k'+2, t)$-paths for S of size 1 can be obtained from the $(k'+1, t)$-paths by starting from each vertex v in G and searching among all its adjacent edges (v, w) to find a $(\{w\}, k'+1, t)$-path to concatenate into a $(\{v\}, k'+2, t)$-path. The above procedures for $t \leq 2$ take $O(m^t)$ time, which is much better than the original $O(n^{2t})$ time for step 2 when G is sparse and also lead to the same improvement for the entire algorithm. Note that since we have $m = O(n^2)$ in the worst case, the worst case time complexity of the algorithm stays the same, but it is much faster in practice for interaction graphs. Also note that this technique is applicable only when $t \leq 2$, since it may take more than t edges in G to specify fully a connection between two sets of vertices of size t when $t \geq 3$.

5. Scoring Path Structures. In reality, for given k and t, there may be many (k, t)-paths in a biological network. We need to assign a score to each (k, t)-path so that biologically relevant structures are likely to get better scores. Since we are interested in identifying paths between two proteins in G, we assume that an additional source and sink are given in addition to G. Since the proteins that are related to the source and the sink in a biological function are more likely to belong to the same pathway, we assign a weight to each vertex in G to reflect its functional relatedness to the source and the sink. For yeast protein interaction networks, we estimate the functional relatedness of proteins by using a compendium set of expression profiles corresponding to 300 mutations and chemical treatments from Hughes et al. [16]. We define the functional similarity $s(p_1, p_2)$ between two proteins p_1 and p_2 to be the Pearson correlation coefficient ρ of the log(expression ratio) across experiments, which is given by

$$\rho = \left(\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})\right) \Bigg/ \sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2},$$

where n is the number of experiments, x_i and y_i are the log(expression ratio) of proteins p_1 and p_2 respectively in the ith experiment, and $\overline{x}$ and $\overline{y}$ are the means of x_i and y_i respectively. The weight of a vertex v in G is defined to be $s(v, \text{source}) + s(v, \text{sink})$, with a higher value representing better functional relatedness to the source and the sink. We then define the score of a (k, t)-path to be the sum of the weight of its k vertices. This allows different (k, t)-paths to be compared directly for a fixed (k, t) pair.

To accommodate the source and the sink, we make the following changes to our algorithm. In step 2.1 ($k \leq t$ and $P_0 = \emptyset$) only return (S, k, t)-paths with the other end connected to the source. After the algorithm is completed, only consider the best (S, k, t)-paths with S connected to the sink. Instead of storing an arbitrary (S, k, t)-path, we remember one currently best (S, k, t)-path with the highest score in steps 2.1, 2.4, and 3.7. Since only vertex weights are used, merging two disjoint optimal path structures always gives a larger optimal path structure. Thus the optimal substructure property is satisfied for our divide-and-conquer technique and the probabilistic guarantee in Theorem 3 is valid for optimal (S, k, t)-paths.

One problem of the current algorithm is that only one (S, k, t)-path is stored for each fixed S within each iteration, and thus not many distinct path structures may be obtained from one run. Since many relevant biological path structures may have suboptimal scores, it would be desirable to return as many distinct path structures as possible in one run. Without significantly increasing the time and space complexity, we modify our algorithm to store top x suboptimal (S, k, t)-paths for each S in steps 2.4 and 3.7. A similar argument shows that a probabilistically guaranteed performance can be obtained for each of the top x suboptimal path structures and the time complexity increases by at most a factor of x^2. To verify that these path structures are sufficiently different from random path structures, we assume that the weights of each vertex are normally distributed and check that the score of a (k, t)-path is at least a few standard deviations away from the mean score of random (k, t)-paths in which k vertices are chosen independently.

6. Application to Protein Interaction Networks.

We test our algorithm on the core protein interaction network of yeast [17] from the DIP database [14], which has 3170 vertices and 6600 edges. Figures 3 and 4 show top scoring path structures on a few pairs of source and sink that correspond to the endpoints of four mitogen-activated protein kinase (MAPK) cascades in [18], including the pheromone response pathway, the filamentation/invasion pathway, the cell integrity pathway, and the high osmolarity pathway, in which detailed biological models are available. The main characteristics of these pathways are that each of them contains a MAPKK kinase, a MAPK kinase, and a MAP kinase in series [18]. To validate our results further, we compare our results with the computational results in [4] and [11]. With $t = 2$, it takes minutes to obtain the results for $k = 10$ from one run of the algorithm, hours for $k = 11$, and many hours to a day for $k = 12$. It takes only slightly more time to guarantee a high probability of finding more than one top suboptimal path structure when compared with finding only the optimal path structure.

In general, a larger value of k results in more interacting proteins found within a path structure and also provides more information. Note that in order to investigate multiple paths, all the results in [4] and some of the results in [11] are aggregates of many linear paths. Since our path structure allows multiple paths in its definition, we can investigate each path structure separately. Due to the relatively small values of k we used, each of our path structures generally contains a smaller number of proteins than their aggregate results. However, our algorithm makes it easier to identify related proteins at the same level of the path structure that may indicate that they play similar roles rather than looking at aggregates of simple paths.

Figure 3 shows the top path structures obtained between Ste3 and Ste12 that correspond to the pheromone response pathway and between Ras2 and Ste12 that correspond to the filamentation/invasion pathway. On the test between Ste3 and Ste12 for the pheromone response pathway, most proteins in the main chain in [18], including Ste3, Ste18, Ste4, Ste5, Ste7, Fus3, and Ste12, were found. A few other proteins that are present in the biological model in [18], including Far1, Bem1, Cdc42, and Gpa1, were

Between Ste3 and Ste12 (pheromone response):

$k = 10$: Ste3—Akr1—Ste18—Ste4—Ste5—Fus3—Gpa1—Sst2—Kss1—Ste12
Ste3—Akr1—(Ste18,Ste5)—(Ste4,Fus3)—Gpa1—Sst2—Kss1—Ste12

$k = 11$: Ste3—Akr1—Ste4—Far1—Bem1—Ste5—Fus3—Gpa1—Sst2—Kss1—Ste12
Ste3—Akr1—Ste18—Ste4—Ste5—Ste7—Fus3—Gpa1—Sst2—Kss1—Ste12

$k = 12$: Ste3—Akr1—Ste18—Ste4—Far1—Bem1—Ste5—Fus3—Gpa1—Sst2—Kss1—Ste12
Ste3—Akr1—Ste4—Far1—Cdc42—Bem1—Ste5—Fus3—Gpa1—Sst2—Kss1—Ste12

Between Ras2 and Ste12 (filamentation/invasion):

$k = 10$: Ras2—Ras1—Cdc25—Ssa2—Gpa1—(Ste4,Fus3)—Ste5—Kss1—Ste12
Ras2—Cdc25—Ssa2—Gpa1—Ste4—Ste5—Fus3—Ste7—Kss1—Ste12

$k = 11$: Ras2—Cdc25—Ssa2—Gpa1—Ste4—Ste5—Fus3—Mpt5—Sst2—Kss1—Ste12
Ras2—Cdc25—Hsp82—Skp1—Cln1—Far1—Ste4—Gpa1—Sst2—Kss1—Ste12

$k = 12$: Ras2—Cyr1—Dbf2—Sec27—Prp11—Ssk2—Far1—Ste4—Gpa1—Sst2—Kss1—Ste12
Ras2—Cyr1—Srv2—Abp1—Cla4—Cdc42—Far1—Ste4—Gpa1—Sst2—Kss1—Ste12

Fig. 3. Top two path structures from the find-paths algorithm on the core protein interaction network of yeast between two pairs of source and sink with k from 10 to 12 and $t = 2$. Two path structures with the same end are stored within the algorithm and the algorithm is run seven times to guarantee a 99.9% probability of finding each of the top two path structures. Proteins within the same level in a path structure are enclosed in parentheses while adjacent levels are connected by a horizontal line.

also found. The proteins Akr1 and Kss1 are also present in the results in [4] and [11], while the protein Sst2 is also present in the results in [4]. Akr1 and Sst2 are both related to the pathway, while Kss1 is related to the almost identical filamentation/invasion pathway that replaces Fus3 by Kss1 [18]. On the test between Ras2 and Ste12 for the filamentation/invasion pathway, our algorithm identified many of the same proteins as above due to the strong resemblance of the two pathways. The proteins Cdc25 and Hsp82 are also present in the results in [4] and [11], while the proteins Cyr1 and Srv2 are also present in the results in [4]. Some number of other proteins were also included that may not be related to the pathway.

Figure 4 shows the top path structures obtained between Rho1 and Rlm1 that correspond to the cell integrity pathway and between Sln1 and Hog1 that correspond to the high osmolarity pathway. Since these pathways are short [18], we set $k = 6$ (we were not able to obtain good results for larger k). On the test between Rho1 and Rlm1 for the cell integrity pathway, the sixth suboptimal path structure that includes the proteins Rho1,

Between Rho1 and Rlm1 (cell integrity):	Between Sln1 and Hog1 (high osmolarity):
Rho1—Rga1—Cdc28—Swi6—Slt2—Rlm1	Sln1—Rvs167—Abp1—Tup1—Sko1—Hog1
Rho1—Rga1—Cdc28—Spa2—Slt2—Rlm1	Sln1—Ypd1—Ssk1—Ssk2—Pbs2—Hog1
Rho1—Rga1—Cdc28—Kin2—Slt2—Rlm1	Sln1—Rvs167—Rxt1—Cyc8—Sko1—Hog1
Rho1—Bni1—Myo3—Bck1—Slt2—Rlm1	Sln1—Rvs167—Idh1—Slt2—Ptp2—Hog1
Rho1—Bni1—Bck1—Mkk2—Slt2—Rlm1	Sln1—Rvs167—Lys12—Slt2—Ptp2—Hog1
Rho1—Pkc1—Mkk1—Bck1—Slt2—Rlm1	Sln1—Rvs167—Idh1—Slt2—Ptp3—Hog1

Fig. 4. Top six path structures from the find-paths algorithm on the core protein interaction network of yeast between two pairs of source and sink with $k = 6$ and $t = 2$. Six path structures with the same end are stored within the algorithm and the algorithm is run seven times to guarantee a 99.9% probability of finding each of the top six path structures.

Pkc1, Mkk1, Bck1, Slt2 and Rlm1 had the best agreement with the biological model in [18], which contains all the components in the main chain. The fifth suboptimal path structure that includes the proteins Rho1, Bni1, Bck1, Mkk2, Slt2, and Rlm1 also agreed extremely well, which includes the protein Bni1 that is related to the pathway. Steffen et al. [4] used a different path length seven to find the pathway, while Scott et al. [11] missed the MAPKK kinase Bck1. On the test between Sln1 and Hog1 for the high osmolarity pathway, the second suboptimal path structure that includes the proteins Sln1, Ypd1, Ssk1, Ssk2, Pbs2, and Hog1 had the best agreement with the biological model in [18], which contains all the components in the main chain. Steffen et al. [4] were unable to find the pathway, while Scott et al. [11] only found the pathway among lower ranked suboptimal paths.

7. Discussion. We have developed a new formulation that allows direct modeling of biological pathways by a non-linear path structure and we gave an algorithm that guarantees that the top suboptimal path structures are found with high probability. Our approach can be easily adapted to analyze directed graphs and it can be applied to other types of biological networks such as metabolic networks. It is also possible to apply the algorithm to analyze conserved path structures in multiple graphs, by following a similar idea as in [5], [9] and [10] first to construct a combined graph from the given graphs to represent desirable vertex correspondences, and then to identify high scoring path structures in the combined graph to obtain alignments of the conserved structures.

One limitation of our current model is that all paths within a path structure must be of the same length l. Although other edges can exist outside the path structure that allow some of the paths to have different lengths and different path structures can have different path lengths, it would be desirable to extend the model to allow multiple paths of different lengths to be included in a path structure directly. Another difficulty is that since the algorithm still has exponential time complexity, it can only be used when k and t are not large. Although our algorithm is asymptotically faster than the previous approaches, for these relatively small values of k and t, we did not observe significant improvements in actual running time when compared with the previous approaches and most of the path structures found were almost linear.

Since the false positive rate of edges in protein interaction networks can be quite high, it may be desirable to take edge reliability also into account [17], [19], [4], [11]. One way to do this is to incorporate edge weights in addition to vertex weights. Although no modifications to the algorithm are necessary since the optimal substructure property is still satisfied, one has to make sure that it is still meaningful to compare directly the scores of two path structures that have different numbers of edges when both vertex and edge weights are used.

References

[1] T. Dandekar, S. Schuster, B. Snel, M. Huynen, and P. Bork, Pathway alignment: application to the comparative analysis of glycolytic enzymes, *Biochem. J.*, **343** (1999), 115–124.

[2] H. Ogata, W. Fujibuchi, S. Goto, and M. Kanehisa, A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters, *Nucleic Acids Res.*, **28** (2000), 4021–4028.

[3] Y. Tohsato, H. Matsuda, and A. Hashimoto, A multiple alignment algorithm for metabolic pathway analysis using enzyme hierarchy, *Proc. 8th Int. Conf. Intell. Systems Mol. Biol.* (*ISMB* 2000), pp. 376–383.

[4] M. Steffen, A. Petti, J. Aach, P. D'haeseleer, and G. Church, Automated modelling of signal transduction networks, *BMC Bioinformatics*, **3** (2002), 34.

[5] B. P. Kelley, R. Sharan, R. M. Karp, T. Sittler, D. E. Root, B. R. Stockwell, and T. Ideker, Conserved pathways within bacteria and yeast as revealed by global protein network alignment, *Proc. Natl. Acad. Sci. USA*, **100** (2003), 11394–11399.

[6] M. Koyutürk, A. Grama, and W. Szpankowski, An efficient algorithm for detecting frequent subgraphs in biological networks, *Bioinformatics*, **20** (2004), SI200–SI207.

[7] H. Hu, X. Yan, Y. Huang, J. Han, and X. J. Zhou, Mining coherent dense subgraphs across massive biological networks for functional discovery, *Bioinformatics*, **21** (2005), SI213–SI221.

[8] R. Y. Pinter, O. Rokhlenko, E. Yeger-Lotem, and M. Ziv-Ukelson, Alignment of metabolic pathways, *Bioinformatics*, **21** (2005), 3401–3408.

[9] R. Sharan, S. Suthram, R. M. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R. M. Karp, and T. Ideker, Conserved patterns of protein interaction in multiple species, *Proc. Natl. Acad. Sci. USA*, **102** (2005), 1974–1979.

[10] M. Koyutürk, Y. Kim, U. Topkara, S. Subramaniam, W. Szpankowski, and A. Grama, Pairwise alignment of protein interaction networks, *J. Comput. Biol.*, **13** (2006), 182–199.

[11] J. Scott, T. Ideker, R. M. Karp, and R. Sharan, Efficient algorithms for detecting signaling pathways in protein interaction networks, *J. Comput. Biol.*, **13** (2006), 133–144.

[12] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

[13] N. Alon, R. Yuster, and U. Zwick, Color-coding, *J. ACM*, **42** (1995), 844–856.

[14] I. Xenarios, D. W. Rice, L. Salwinski, M. K. Baron, E. M. Marcotte, and D. Eisenberg, DIP: the Database of Interacting Proteins, *Nucleic Acids Res.*, **28** (2000), 289–291.

[15] H. L. Bodlaender, A tourist guide through treewidth, *Acta Cybernet.*, **11** (1993), 1–21.

[16] T. R. Hughes, M. J. Marton, A. R. Jones, C. J. Roberts, R. Stoughton, C. D. Armour, H. A. Bennett, E. Coffey, H. Dai, Y. D. He, M. J. Kidd, A. M. King, M. R. Meyer, D. Slade, P. Y. Lum, S. B. Stepaniants, D. D. Shoemaker, D. Gachotte, K. Chakraburtty, J. Simon, M. Bard, and S. H. Friend, Functional discovery via a compendium of expression profiles, *Cell*, **102** (2000), 109–126.

[17] C. M. Deane, L. Salwinski, I. Xenarios, and D. Eisenberg, Protein interactions: two methods for assessment of the reliability of high throughput observations, *Mol. Cell. Proteomics*, **1** (2002), 349–356.

[18] M. C. Gustin, J. Albertyn, M. Alexander, and K. Davenport, MAP kinase pathways in the yeast *Saccharomyces cerevisiae*, *Microbiol. Mol. Biol. Rev.*, **62** (1998), 1264–1300.

[19] M. Deng, S. Mehta, F. Sun, and T. Chen, Inferring domain–domain interactions from protein–protein interactions, *Genome Res.*, **12** (2002), 1540–1548.