Finding Supported Paths in Heterogeneous Networks

Guillaume Fertin, Christian Komusiewicz, Hafedh Mohamed-Babou, Irena Rusu

Algorithms **2015**, *8*, 810-831; doi:10.3390/a8040810

OPEN ACCESS

algorithms

ISSN 1999-4893

www.mdpi.com/journal/algorithms

Article

Finding Supported Paths in Heterogeneous Networks †

Guillaume Fertin [1,*], Christian Komusiewicz [2], Hafedh Mohamed-Babou [1] and Irena Rusu [1]

[1] LINA, UMR CNRS 6241, Université de Nantes, Nantes 44322, France;
E-Mails: hafedh.mohamed-babou@univ-nantes.fr (H.M.-B.); irena.rusu@univ-nantes.fr (I.R.)

[2] Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Berlin D-10587, Germany; E-Mail: christian.komusiewicz@tu-berlin.de

† This paper is an extended version of our paper published in the Proceedings of the 11th International Symposium on Experimental Algorithms (SEA 2012), Bordeaux, France, 7–9 June 2012, originally entitled "Algorithms for Subnetwork Mining in Heterogeneous Networks".

* Author to whom correspondence should be addressed; E-Mail: guillaume.fertin@univ-nantes.fr; Tel.: +33-2-5112-5824.

Academic Editor: Giuseppe Lancia

Received: 17 June 2015 / Accepted: 29 September 2015 / Published: 9 October 2015

Abstract: Subnetwork mining is an essential issue in the analysis of biological, social and communication networks. Recent applications require the simultaneous mining of several networks on the same or a similar vertex set. That is, one searches for subnetworks fulfilling different properties in each input network. We study the case that the input consists of a directed graph D and an undirected graph G on the same vertex set, and the sought pattern is a path P in D whose vertex set induces a connected subgraph of G. In this context, three concrete problems arise, depending on whether the existence of P is questioned or whether the length of P is to be optimized: in that case, one can search for a longest path or (maybe less intuitively) a shortest one. These problems have immediate applications in biological networks and predictable applications in social, information and communication networks. We study the classic and parameterized complexity of the problem, thus identifying polynomial and NP-complete cases, as well as fixed-parameter tractable and W[1]-hard cases. We also propose two enumeration algorithms that we evaluate on synthetic and biological data.

Keywords: NP-hard problems; directed acyclic graphs; longest path problem; shortest path problem; protein interaction networks; metabolic networks

1. Introduction

The use of social and telecommunication networks has dramatically increased recently, resulting in new prominent applications of network analysis. In addition to these real-world applications, network representations of new types of data, in particular biological data, highlight the drastic need for a new, multi-dimensional, type of (sub)network mining in which several networks, representing several relations between the same objects, are simultaneously investigated for the extraction of a multi-dimensional pattern [1–3].

The study of multi-dimensional mining started several years ago, but it mainly concerns homogeneous representations of data: directed graph alignment [4], undirected graph alignment [5], relational data mining [6] and social networks mining [2] are several examples. Recently, such approaches found applications in computational biology [7–9], but also showed their limits, due to the multiple types of biological networks that are used to describe different views of the same biological process. In such applications, a process is often represented as a path in a directed network, for example a metabolic network [9–11] or a signaling network [12,13], and as a connected graph in an undirected network, for example a protein-protein interaction network [14]; the two networks are linked by the components involved in the process, which are represented as vertices in each network. Identifying a particular biological process then requires identifying parts of the two networks (directed and undirected) that have the desired topological patterns and the same vertex set. This multi-dimensional mining approach has led to the discovery of novel biological insights [3,15–17].

In this paper, we consider a network mining across two heterogeneous networks, driven by the previously-cited applications in biological networks. Following the situation described above, we consider the case in which we are given a directed graph D and an undirected graph G and want to find a path P in D (that is, a graph with vertex set $V(P) = \{p_1, \ldots, p_k, p_{k+1}\}$, $k \geq 1$, and arc set $\{(p_i, p_{i+1}) : 1 \leq i \leq k\}$), whose vertices induce a connected subgraph in G. In the application context when D is a metabolic network and G is a protein interaction network, such a problem corresponds to searching for a chain of reactions (a path in D) involving proteins that form a complex (that is connected in G).

Definition 1. *Let $D = (V, A)$ be a directed graph and $G = (V, E)$ be an undirected graph on the same vertex set. A (D, G)-supported path is a simple (directed) path P of length at least one in D, such that the subgraph of G induced by $V(P)$, denoted $G[V(P)]$, is connected.*

The length restriction in the definition simply allows us to disregard trivial (D, G)-supported paths consisting of only one vertex. We consider several natural variants of finding (D, G)-supported paths. The most basic problem simply asks for the existence of such a path.

SUPPORTED PATH (SP)
Instance: A directed graph $D = (V, A)$ and an undirected graph $G = (V, E)$.
Question: Is there a (D, G)-supported path?

In applications, one might additionally try to optimize certain criteria of such a path. We study here the fundamental problems of maximizing or minimizing path length:

LONGEST SUPPORTED PATH (LSP)
Instance: A directed graph $D = (V, A)$ and an undirected graph $G = (V, E)$.
Task: Find the longest (D, G)-supported path.

SHORTEST SUPPORTED PATH (SSP)
Instance: A directed graph $D = (V, A)$ and an undirected graph $G = (V, E)$.
Task: Find the shortest (D, G)-supported path.

In this work, we study the complexity landscape of these computational problems by identifying polynomial-time solvable, as well as NP-hard cases of the problem. Clearly, more general variants of the problem, for example when D and G have different, but related vertex sets, may be also useful in practice, but their study is beyond the scope of this work.

Before presenting our main results, we make some basic observations on the complexity of the problems. First, with respect to polynomial-time solvability, SP is clearly not harder than SSP and LSP. Moreover, consider the special case in which G is a clique. This simply means that the graph G does not put any additional constraints on the desired paths. In this case, SP and SSP reduce to the polynomial-time problem of deciding whether D has at least one arc, but LSP is the NP-hard LONGEST PATH path problem. Therefore, intuitively, LSP is the hardest of the three problems. Note that for LSP, the special case in which the directed graph D is a DAG and in which G is a clique is polynomial-time solvable, since it reduces to finding the longest path in a DAG.

Preliminaries. Throughout the paper, D will denote a directed graph and G an undirected graph, built on the same vertex set V. We use $n := |V|$ to denote the number of vertices in D (or equivalently, in G). Given a graph H and a set $S \subseteq V(H)$, let $H[S]$ denote the subgraph of H induced by S, that is the graph with vertex set S and all edges (arcs) of H that have both endpoints in S. We use $N_H(v)$ to denote the neighborhood of a vertex v in a graph H and $N_H[v] := N_H(v) \cup \{v\}$ to denote the closed neighborhood of v in H. A path P is called simple if it contains every vertex of $V(P)$ exactly once.

A star is a tree with one non-leaf, a bi-star is a tree with two non-leafs. The diameter of a graph is the maximum length of a shortest path between any two of its vertices. An outerplanar graph is a graph admitting a planar embedding with all vertices on a circle, all edges inside the circle and such that edges do not cross each other.

A problem with input size n is called fixed-parameter tractable with respect to a parameter k if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function only depending on k. A fundamental class of presumed parameterized intractability is called W[1]. Hardness for W[1] can be shown by a parameterized reduction from a problem that is known to be W[1]-hard. A parameterized reduction transforms an instance (X, k) of a parameterized problem L in $f(k) \cdot |X|^{O(1)}$ time into an equivalent

instance (X', k') of a parameterized problem L'. We refer to [18–20] for further details concerning parameterized complexity.

Our Results. Our findings for the three problems can be summarized as follows (see Table 1 for an overview, where h is the number of vertices of degree two or more). Here, for any directed graph H, the underlying undirected graph of H, denoted H^*, is the graph that is obtained from H simply by removing arc directions.

Table 1. Complexity of the variants of SUPPORTED PATH. Herein, D^* is the underlying undirected graph of D. If no problem names are specified, then the results hold for SP, LONGEST SUPPORTED PATH (LSP)and SHORTEST SUPPORTED PATH (SSP); arrows indicate that a result is implied by another result in the table.

G	Acyclic D			General D
	D^*			
	Tree	**Outerplanar**	**Treewidth 3**	
path or cycle	P (Proposition 2)			NPC [21]
h vertices of degree ≥ 2	P $\downarrow$	$2^h \cdot n^{O(1)}$ (Proposition 2)		LSP, $h = 1$: NPC (Theorem 2) SSP, $h = 2$: P (Proposition 1) SSP, SP, $h = 3$ and G is a tree: NPC (Theorem 1)
tree with diameter 4	P $\downarrow$	LSP: NPC (Theorem 3)	NPC (Theorem 4)	NPC $\uparrow$
general graph	P (Proposition 2)	LSP: NPC $\uparrow$	NPC $\uparrow$	NPC $\uparrow$

If the underlying undirected graph D^* of D is a tree, then all three problems can be solved in polynomial time. If D^* is outerplanar, then LSP is already NP-hard. If D^* has treewidth three, then even SP is NP-complete. These hardness results hold even if D is acyclic and G is a tree with diameter four. For general digraphs, all problems become hard even if G is a tree with a constant number, at least three, of non-leaf vertices. In contrast, if D is acyclic, then all three problems become fixed-parameter tractable with respect to the number of vertices with degree at least two in G. These general complexity results are presented in Section 2 (which considers general digraphs) and Section 3 (which considers the special case where D is acyclic).

In Section 4, we then study the parameterized complexity of the problems, showing that deciding whether G contains a (D, G)-supported path of length exactly k is W[1]-hard, that is a fixed-parameter algorithm for this parameter is unlikely. Thus, a running time of $n^{f(k)}$ seems to be more or less unavoidable for finding (D, G)-supported paths of length k. This holds even if G is a tree and D is acyclic.

In Section 5, we propose two enumeration-based algorithms for solving the problem of finding (D, G)-supported paths of length exactly k. One algorithm is a fixed-parameter algorithm for the combined parameter $(\tilde{\Delta}, k)$, where $\tilde{\Delta}$ is the minimum of the maximum outdegree and maximum indegree in D. The other algorithm is a fixed-parameter algorithm for the combined parameter (Δ, k), where Δ is the maximum degree in G. Furthermore, we describe in Section 6 several data reduction rules that are used in the implementation of our algorithms.

In Section 7, we evaluate our two enumeration-based algorithms on synthetic and biological data. Our main observations are that finding (D, G)-supported paths with at most eight vertices is tractable in the considered biological network and that the case in which G is very sparse compared to D is hard to handle for our algorithm. This is in line with our theoretical findings that show that all three problem variants remain hard if G is a tree. In Section 8, we conclude with a summary of our results and directions for future research.

2. Complexity Classification for General Digraphs

In this section, we study the complexity of SP, LSP and SSP considering the most general case for the directed graph while putting very strong restrictions on the undirected graph G. Our main result here is the following hardness result.

Theorem 1. SUPPORTED PATH *is NP-hard even if G is a tree with two vertices that have degree at least three and one degree-two vertex.*

Proof. We describe a reduction from the following NP-hard problem (see also Figure 1 for an illustration).

> DIRECTED DISJOINT PATHS
> Instance: A directed graph $H = (W, F)$, and k vertex pairs (s_i, t_i), $1 \leq i \leq k$.
> Question: Is there a set of vertex-disjoint paths P_i, each going from s_i to t_i?

DIRECTED DISJOINT PATHS are NP-hard even if $k = 2$ [22]. Let $(H, 2)$ be an instance of DIRECTED DISJOINT PATHS. Assume, without loss of generality, that s_1 and s_2 have only outgoing incident arcs and t_1 and t_2 have only incoming incident arcs. Furthermore, assume that t_1 is not a neighbor of s_1 (and note that it cannot be a neighbor of t_2 by the assumption above). Furthermore, assume that $N_H[s_1]$ and $N_H[t_2]$ are disjoint. This can be easily achieved for example by subdividing the arcs incident with s_1. Construct an instance of SUPPORTED PATH as follows. First, initialize the directed graph as $D := H$ and add the arc (t_1, s_2) to D. Note that s_2 is the only outneighbor of t_1 and t_1 is the only inneighbor of s_2. Then, let G be the tree on W with the following properties:

- t_1 is a neighbor of t_2 and s_1,
- all neighbors of s_1 in D are leaf neighbors of t_2 and
- s_2 and all other vertices of D are leaf neighbors of s_1.

Note that, by definition, in G, all vertices are leaves, except t_1, which has degree two, and s_1 and t_2, which have an arbitrarily large degree. We now show the equivalence of the instances, that is H has two vertex-disjoint paths from s_1 to t_1 and from s_2 to t_2 if and only if there exists a (D, G)-supported path.

$\Rightarrow$: Let P_1 and P_2 be two vertex-disjoint paths in H from s_1 to t_1 and from s_2 to t_2, respectively. Then, the concatenation of P_1 and P_2 is a simple directed path in D, since D additionally contains the arc (t_1, s_2). Further, $G[V(P_1) \cup V(P_2)]$ is connected, since all vertices that are in $W \setminus (V(P_1) \cup V(P_2))$ have degree one in G, and thus, their deletion from G cannot make G disconnected.

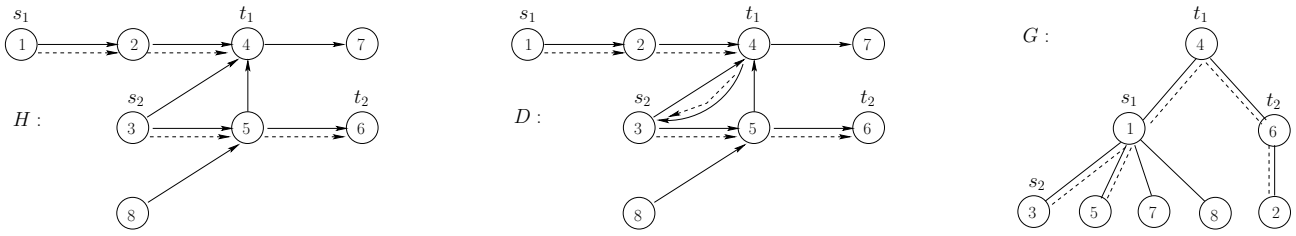

Figure 1. Illustration of the reduction in the proof in the proof of Theorem 1. The arcs/edges of H, D and G are drawn with plain lines. The dotted lines represent, respectively: in H, the paths joining s_1 to t_1 and s_2 to t_2; in D, the corresponding (D, G)-supported path; in G, the connected subgraph given by the vertices of the (D, G)-supported path.

$\Leftarrow$: Let P be a (D, G)-supported path. Every edge in G has one of its endpoints in $\{s_1, t_2\}$. Hence, P contains either s_1 or t_2. Since P contains at least two nodes, then it contains, in D, a neighbor of s_1 or a neighbor of s_2. All neighbors of s_1 in D are leaf neighbors of t_2 in G, and all neighbors of t_2 in D are leaf neighbors of s_1 in G (recall that we assume $N_H(t_2)$ to be disjoint from $N_H(s_1)$). Consequently, P contains s_1 and t_2 and, thus, also t_1. Since s_1 has no incoming arcs, P starts in s_1, and since t_2 has no outgoing arcs, P ends in t_2. Consequently, P contains an outneighbor of t_1, which by construction is s_2. Hence, P is a simple path in D from s_1 to t_1 to s_2 to t_2. Hence, there are two vertex-disjoint paths P_1 and P_2 from s_1 to t_1 and from s_2 to t_2 in H. □

Hence, to obtain tractable cases for general D, one would need to constrain G even further. If G is a bi-star, SHORTEST SUPPORTED PATH and, thus, also SUPPORTED PATH, are easily solvable in polynomial time.

Proposition 1. SHORTEST SUPPORTED PATH *can be solved in polynomial time if G is a bi-star.*

Proof. Let G be a bi-star with non-leaf vertices u and v. Every (D, G)-supported path P thus contains either u or v. In the case where $D[N_G[u]]$ contains an arc incident with u, then there is a (D, G)-supported path of length one containing u, and we are done. Similarly, in the case where $D[N_G[v]]$ contains an arc incident with v, then there is a (D, G)-supported path of length one containing v.

Let us now consider the case where $D[N_G[u]]$ does not contain an arc incident with u, and $D[N_G[v]]$ contain no arc incident with v. Let P be a (D, G)-supported path (if any). Thus, P^* (the underlying undirected path of P) contains no edge, which is also present in G. Consequently, P links a neighbor of v to u, or a neighbor of u to v, or a neighbor of v to a neighbor of u, or a neighbor of u to a neighbor of v. If u or v does not belong to P, then $G[V(P)]$ cannot be connected. Thus, P contains u and v. Further, since G is a bi-star with non-leaf vertices u and v, every path in D containing u and v is supported. Hence, the problem can be solved by computing the shortest path between u and v in this case (considering both possible directions of the path). If such a path does not exist, then there is no (D, G)-supported path. □

The LONGEST SUPPORTED PATH problem is more difficult, since it remains NP-hard even if G is a star: a simple reduction from the NP-hard HAMILTONIAN PATH problem, which asks for a simple path on all vertices of a directed graph D, can be achieved by making G an arbitrary star on the same vertex set as the input graph D of the HAMILTONIAN PATH instance. Then, by finding the longest (D, G)-supported path, one can clearly solve the HAMILTONIAN PATH.

Theorem 2. *The decision version of* LONGEST SUPPORTED PATH *is NP-hard, even if G is a star.*

A similar hardness result for LONGEST SUPPORTED PATH can be easily obtained if G is a path and apparently even SUPPORTED PATH is NP-hard in this case [21].

3. Complexity Classification for Acyclic Digraphs

In this section, we continue our complexity analysis for the special case where D is acyclic. First, we identify some special cases in which all three problems can be solved in polynomial time.

Proposition 2. *If D is acyclic, then* LONGEST SUPPORTED PATH *and* SHORTEST SUPPORTED PATH *can be:*

(a) solved in polynomial time if D^* is a tree,
(b) solved in polynomial time if G is a chordless path or cycle or
(c) solved in $2^k \cdot n^{O(1)}$ time if G has at most k vertices with degree two or more.

Proof. Before studying each case, we note that, in a DAG, the longest path, as well as the longest path between two given vertices can be computed in linear time, using dynamic programming.

(a) Since D^* is a tree, the set $\mathcal{P}$ of (directed) paths in D can be computed in polynomial time: there is at most one path between every pair of vertices in D, and if it exists, it can be found in linear time for each pair of endpoints. Hence, the longest and the shortest (D, G)-supported path can be found as follows. Check for each pair of endpoints u and v whether there is a path P between u and v in D. If this is the case, check whether $G[P]$ is connected. If this is the case, keep P if it is longer or shorter than the currently stored shortest or longest paths.

(b) Since G is a path or a cycle, the number of connected subgraphs of G is $O(n^2)$, and they may be computed in polynomial time. Further, for each connected subgraph $G' = (V', E')$ of them, one can compute in linear time whether $D[V']$ has a Hamiltonian path, because D is acyclic. The longest (resp. shortest) (D, G)-supported path is the maximum-order (resp. minimum-order) subgraph G' of G for which this is the case.

(c) Let X denote the set of k vertices with degree two or more in G. For each $X' \subseteq X$, we solve the problem of finding a (D, G)-supported path that contains X' as follows. Clearly, we only need to consider X', such that $G[X']$ is connected. Further, we can remove from G all degree-one vertices that have no neighbor in X'. Let Y denote the set of remaining degree-one vertices, and note that $G[X' \cup Y']$ is connected for all $Y' \subseteq Y$. Hence, it is sufficient to compute the longest (resp. shortest) path in $D[X' \cup Y]$ that contains all vertices of X'. Fix an arbitrary topological ordering of $D[X' \cup Y]$. Then, the order $(x_1, x_2, \ldots, x_{|X'|})$ of the vertices of X' in this topological ordering determines the order of the vertices of $D[X' \cup Y]$ in any path that contains X': by definition of topological orderings, there is no directed path from x_{i+1} to x_i, so x_{i+1} must be after x_i in such a path. By the same argument, for each x_i, $1 \leq i < |X'|$, any path between x_i and x_{i+1} contains only vertices that are put between x_i and x_{i+1} by this topological ordering. Hence, the longest (resp. shortest) path that contains X' in this graph can be computed by concatenating the longest (resp. shortest) paths between x_i and x_{i+1} for all i, $1 \leq i < |X'|$. As discussed above, each such path can be computed in polynomial time. Hence, the overall running time is $2^k \cdot n^{O(1)}$.

□

Note that Case (c) includes the cases in which G is a star or a bi-star, that is, all trees with diameter at most three. To extend these tractability results, one might try, for example, to achieve polynomial-time algorithms for the case that D^* and G have bounded treewidth or for the case that G has diameter four. The theorem below, however, shows that LONGEST SUPPORTED PATH becomes NP-hard in these cases.

Theorem 3. *The decision version of* LONGEST SUPPORTED PATH *is NP-complete, even when D is acyclic, D^* is an outerplanar graph and G is a tree with diameter four.*

Proof. The problem is clearly in NP. To show NP-hardness, we describe a reduction from MAX 2-SAT [23] (see also Figure 2 for an illustration of the reduction). Let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_p$ be an instance of MAX 2-SAT, built from a collection $\mathcal{C} = \{C_1, \ldots, C_p\}$ of p clauses with two literals each, over the variable set $\mathcal{X}_n = \{x_1, \ldots x_n\}$. Let D be built on $2p + 2n + 2$ levels, called optional (marked with a star) or compulsory (not marked):

- Level 0: a vertex s;
- level* $2i - 1$, $1 \leq i \leq p$: two vertices $v_{i,1}$ and $v_{i,2}$ corresponding to the literals of clause C_i;
- level $2i$, $1 \leq i \leq p$: a vertex c_i corresponding to clause C_i;
- level $2p + 1$: two vertices $v_{p+1,1}$ and $v_{p+1,2}$ corresponding, respectively, to literals x_n and $\overline{x_n}$;
- level $2p + 2$: a vertex c_{p+1};
- level $2p + 2 + 2i - 1$, $1 \leq i \leq n$: two vertices a_i and b_i;
- level $2p + 2 + 2i$, $1 \leq i < n$: a vertex A_i.

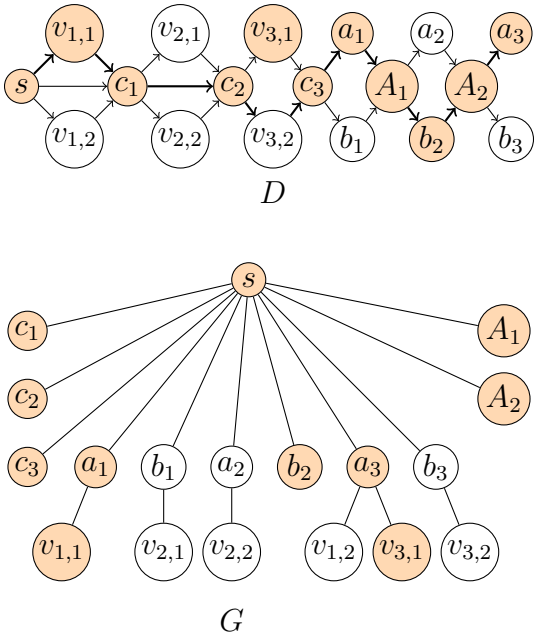

Figure 2. Construction of D and G from the following instance of MAX 2-SAT with $p = 2$ and $n = 3$: $\phi = (x_1 \vee x_3) \wedge (\overline{x_1} \vee x_2)$. Levels in D are represented from left to right. The orange vertices form the (D, G)-supported path corresponding to the assignment $\{x_1 = \text{true}, x_2 = \text{false}, x_3 = \text{true}\}$.

Then, add (a) all possible arcs between any two consecutive levels, (b) the arc (s, c_1) and (c) the arcs $(c_i, c_{i+1}), 1 \leq i < p$. It is clear that D is a DAG. To see that D^* is an outerplanar graph, it is sufficient to

draw the vertices on a circle according to the order $s, v_{1,1}, c_1, v_{2,1}, c_2, \ldots, v_{p+1,1}, c_{p+1}, a_1, A_1, \ldots, a_{n-1}, A_{n-1}, a_n, b_n, \ldots, b_1, v_{p+1,2}, \ldots, v_{1,2}$.

Graph G is a tree with root s. There is an edge between s and each vertex in $\{a_i, b_i : 1 \leq i \leq n\} \cup \{A_i : 1 \leq i < n\} \cup \{c_i : 1 \leq i \leq p+1\}$. There is an edge between each vertex a_i (resp. b_i) and any vertex $v_{\ell,m}$ with $1 \leq \ell \leq p+1, 1 \leq m \leq 2$, such that $v_{\ell,m}$ corresponds to the literal x_i (resp. $\overline{x_i}$). Figure 2 is an illustration of the reduction described above, with $\phi = (x_1 \vee x_3) \wedge (\overline{x_1} \vee x_2)$. Notice that G is of diameter four, where s plays a central role. We now claim that there is an assignment for the variables in $\mathcal{X}_n$ that satisfies at least k clauses if and only if there is a (D, G)-supported path of length at least $p + k + 1 + 2n$.

$\Rightarrow$: Let $\mathcal{A}$ be an assignment of the variable set $\mathcal{X}_n$ that satisfies k' clauses $C_{i_1}, \ldots, C_{i_{k'}}$ of $\mathcal{C}$, where $k' \geq k$. Assume without loss of generality that vertices $v_{i_1,1}, \ldots, v_{i_{k'},1}, v_{p+1,1}$ correspond to true literals. Let $\mathcal{B}(i) = a_i$ if x_i is true, and $\mathcal{B}(i) = b_i$ otherwise. Then, the path P with vertices $s, v_{i_1,1}, \ldots, v_{i_{k'},1}, v_{p+1,1}, c_1, \ldots, c_{p+1}, \mathcal{B}(1), A_1, \mathcal{B}(2), A_2, \ldots, \mathcal{B}(n-1), A_{n-1}, \mathcal{B}(n)$ is (D, G)-supported and has length $p + k' + 1 + 2n$. Indeed, in G, the vertices $v_{i_1,1}, \ldots, v_{i_{k'},1}, v_{p+1,1}$ are connected to s using the corresponding vertex $\mathcal{B}(\ell_{i_j})$ (where $v_{i_j,1}$ is $x_{\ell_{i_j}}$ or $\overline{x_{\ell_{i_j}}}$), $1 \leq j \leq k'$ and $\mathcal{B}(n)$, respectively. All of the other vertices are adjacent to s.

$\Leftarrow$: Let P be a (D, G)-supported path P of length at least $p+k+1+2n$, *i.e.*, with at least $p+k+2n+2$ vertices. Now, there are at most $p+2+2n$ compulsory levels implying that P contains at least k vertices from optional levels. As P cannot contain two vertices associated with literals $x_i, \overline{x_i}$ for some i (because at most one of the vertices a_i and b_i, needed for the connection in G, belong to P), we may safely assign the value $true$ to the literals associated with the vertices in P on optional levels. These literals yield a correct assignment that satisfies k clauses. □

The above reduction shows hardness for LONGEST SUPPORTED PATH. By slightly extending the class of graphs for D^*, we can also show hardness for SUPPORTED PATH and, thus, also for SHORTEST SUPPORTED PATH, even if D is acyclic. This reduction also implies that LONGEST SUPPORTED PATH is not f-approximable, for any function f, even if D is acyclic.

Theorem 4. SUPPORTED PATH *is NP-complete even when G is a tree of diameter four, and the underlying undirected graph D^* of D has treewidth three.*

Proof. The proof is by a reduction from 3-SAT. Let ϕ be a Boolean formula with clauses $C_i = (l_{i,1}, l_{i,2}, l_{i,3})$, $1 \leq i \leq p$ over the set of variables $\mathcal{X}_n = \{x_1, x_2, \ldots, x_n\}$. Assume that $C_p = (x_n, x_n, x_n)$ (this can be achieved by adding one variable and one clause to the input formula ϕ). We now describe how to build D and G from ϕ. The directed acyclic graph D is built on $p+n+1$ levels, numbered from zero to $p + n$. Each level contains between one and three vertices:

1. Level 0: one vertex s,
2. level i, $1 \leq i \leq p$: three vertices $l_{i,1}, l_{i,2}, l_{i,3}$ that correspond to the variables occurring in clause C_i,
3. level $p + j$, $1 \leq j \leq n$: two vertices a_j and b_j.

We complete the construction of D by adding all arcs from any vertex on level i to any vertex on level $i + 1$, for all $0 \leq i < p + n$. The undirected graph G has the same vertex set as D. We define X_j (resp. $\overline{X_j}$) to be the set of vertices in $V(G)$ such that the literal corresponding to this vertex is x_j (resp. $\overline{x_j}$), for all $1 \leq j \leq n$. The edges of G are obtained as follows:

1. put an edge between each vertex in X_j and vertex a_j
2. put an edge between each vertex in $\overline{X_j}$ and vertex b_j
3. put an edge $\{s, a\}$ for each a_i and an edge $\{s, b\}$ for each b_i

Figure 3 gives an illustration of the above reduction. Notice that G is a tree of diameter four, in which s plays a central role. Note also that D^* has treewidth three, since each level consists of three vertices, and all arcs are between consecutive levels. It remains to show the equivalence of the instances, which is formalized as follows: ϕ is satisfiable if and only if there is a (D, G)-supported path of length at least one.

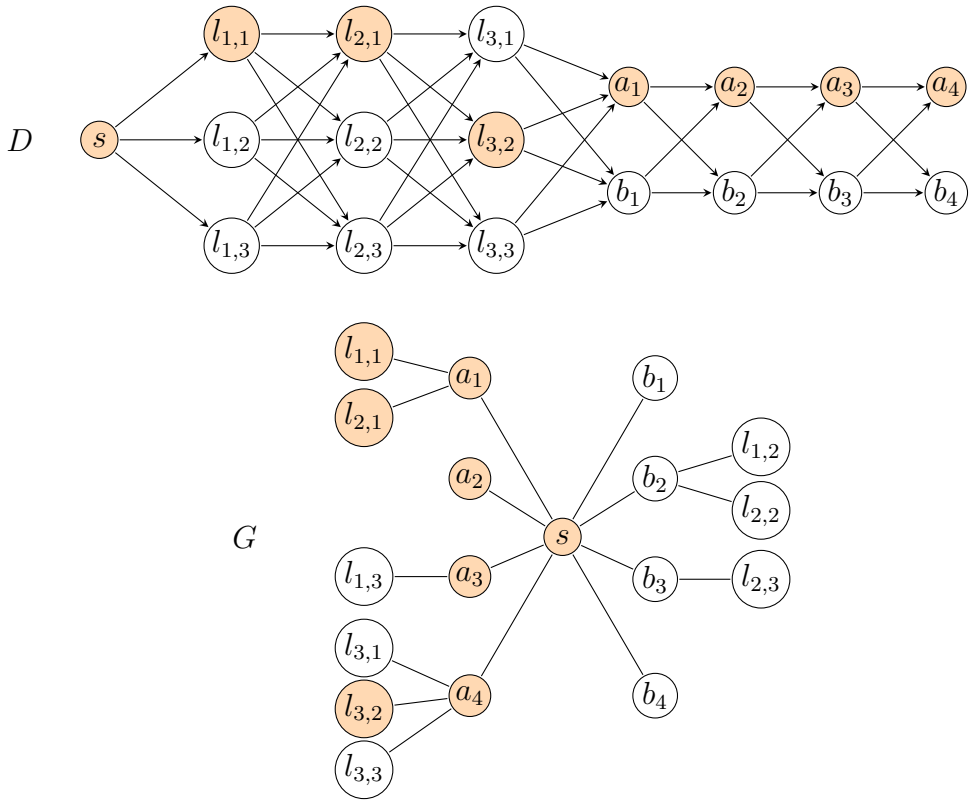

Figure 3. Illustration of the reduction in the proof of Theorem 4. Here, $\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_4 \vee x_4 \vee x_4)$, and the levels are drawn from left to right. The (D, G)-supported path marked in orange corresponds to the assignment $x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{true}, x_4 = \text{true}$.

$\Rightarrow$: Let $\mathcal{A}$ be an assignment of the variables $x_1, \ldots, x_n$ that satisfy ϕ. Consider a path P in D that contains s and for each level i, $1 \leq i \leq p$, one (arbitrary) vertex that corresponds to a literal that is set to true by $\mathcal{A}$, and for each level i, $p < i \leq p + n$, the vertex a_i if $\mathcal{A}$ assigns true to x_i and the vertex b_i, otherwise. We now show that $G[V(P)]$ is connected, which implies that P is a (D, G)-supported path of length at least one.

First, each vertex a_i or b_i is adjacent to s, so the subgraph of $G[V(P)]$ induced by s plus the vertices from levels $i > p$ is connected. Every other vertex $l_{i,j} \in V(P)$ is adjacent to one vertex in this subgraph: Since $l_{i,j}$ satisfies C_i, assignment $\mathcal{A}$ sets the corresponding literal to true. By construction, this means

that if $l_{i,j}$ corresponds to a non-negated variable x_q, then $a_q \in V(P)$ and a_q and $l_{i,j}$ are adjacent in G; otherwise, $b_q \in V(P)$ and b_q and $l_{i,j}$ are adjacent in G.

$\Leftarrow$: We first show that any (D, G)-supported path P of length at least one contains a vertex from each level of D. Every edge in G has one endpoint in a vertex that is in level $i > p$ in D. Now, the other endpoint of this edge is in some level $i \leq p$ in D. Consequently, P contains one of the vertices from layer p in D. These vertices all correspond to the literal x_n, and thus, P contains a_n, since a_n is the only neighbor of these vertices in G. Now, this means that P contains one vertex from all levels $i > p$. Since $G[V(P)]$ is connected, P contains s. Consequently, P contains a vertex from the first and the last level of D, and thus, it contains a vertex from each level of D.

By construction, P contains at most one vertex from each level, and thus, the selected vertices from levels $i > p$ directly define an assignment $\mathcal{A}$ of the variables of ϕ: set variable x_i to true if $a_i \in V(P)$ and to false if $b_i \in V(P)$. Moreover, this assignment satisfies ϕ: Each vertex v in layer i, $1 \leq i \leq p$, corresponds to a literal of C. Since $G[V(P)]$ is connected, this vertex v has a neighbor in $G[V(P)]$. If the literal corresponds to a non-negated variable x_q, then this neighbor is a_q; otherwise, this neighbor is b_q. Hence, assignment $\mathcal{A}$ makes this literal true, and thus, satisfies each clause of ϕ. □

4. Parameterized Hardness

The hardness results presented so far do not exclude efficient algorithms for the case in which one is satisfied with finding a (D, G)-supported path of length exactly k for some given constant k. Such an algorithm might be interesting for example in case the longest path in D has bounded length. In fact, an $n^{k+O(1)}$-time algorithm for this problem is easily achievable by guessing the vertices of the path in the correct order and then checking in polynomial time whether the subgraph of G is connected. As we show in the following, it is unlikely that this can be improved to a running time of $f(k) \cdot n^{O(1)}$.

Theorem 5. *The problem of deciding whether there is a (D, G)-supported path of length exactly k is W[1]-hard even if D is acyclic and G is a tree of diameter four.*

Proof. We describe a parameterized reduction from the W[1]-hard MULTICOLORED CLIQUE problem [24]:

> Instance: An undirected graph $H = (W, F)$ with a proper vertex coloring $c : V \to \{1, \ldots, k\}$.
> Question: Is there a clique S of order k in H such that S contains one vertex from each color?

Given an instance of MULTICOLORED CLIQUE, the instance of SUPPORTED PATH is constructed as follows. The vertex set is $W \cup \bigcup_{\{i,j\} \in F} \{v_{i,j}, v_{j,i}\} \cup \{s\}$, that is we create one vertex for each vertex of H, two vertices for each edge of H and one additional vertex $\{s\}$. We call the two vertices that correspond to the same edge in H siblings. In the construction, assume that an arbitrary, but fixed ordering of unordered color pairs is given.

The directed graph has the following $1 + k^2$ levels:

- Level 0 contains s,
- level i, $1 \leq i \leq k$, contains all vertices of W that have color i,
- level $k + 2i - 1$, $1 \leq i \leq k \cdot (k-1)/2$, contains all vertices $v_{j,\ell}$ where $\{c(j), c(\ell)\}$ is the i-th color pair and $c(j) < c(\ell)$ and
- level $k + 2i$, $1 \leq i \leq k \cdot (k-1)/2$, contains all vertices $v_{\ell,j}$ where $\{c(j), c(\ell)\}$ is the i-th color pair and $c(\ell) > c(j)$.

Now, add arcs in D as follows:

- for level $i \leq k$, add an arc from each vertex in level i to each vertex in level $i + 1$,
- for level $k + 2i - 1$, $1 \leq i \leq k \cdot (k-1)/2$, add an arc from each vertex in level $k + 2i - 1$ to its sibling in level $k + 2i$ and
- for level $k + 2i$, $1 \leq i < k \cdot (k-1)/2$, add an arc from each vertex in level $k + 2i$ to each vertex in level $k + 2i + 1$.

This completes the construction of D. The undirected graph G is now constructed by adding the following edges:

- for each $w \in W$, add $\{s, w\}$ and
- for each $v_{i,j}$, add $\{i, v_{i,j}\}$.

Informally, the idea of the construction is to force with D that each supported path of the correct length selects exactly one vertex of each color and two vertices for each color pair. With the arcs in D between levels $k + 2i - 1$ and $k + 2i$, we force that two vertices of the same color pair that are selected correspond to an edge in H. Finally, with the construction of G, we force that a vertex $v_{i,j}$ of a color pair can only be selected if its endpoint i is selected. To complete the formal correctness proof of our parameterized reduction, we show the equivalence of the instances.

(H, c) is a yes instance if and only if there is a (D, G)-supported path of length exactly k^2.

$\Rightarrow$: Let S be a clique of order k in H. Since c is a proper vertex coloring, S contains exactly one vertex of each color. Now, consider the vertex set V_P that contains $\{s\}$, S, and all vertices corresponding to an edge of $H[S]$. First, note that $|V_P| = 1 + k + 2\binom{k}{2} = 1 + k^2$. Second, $G[V_P]$ is connected, since $G[S \cup \{s\}]$ is a star, and every vertex in $V_P \setminus (S \cup \{s\})$ is in G a neighbor of one of its endpoints in V_P. Finally, $D[V_P]$ is an induced path of length $1 + k^2$, which can be seen as follows. All vertices are on different levels of D. For each $i \leq k$, the vertex of V_P on level i has an arc to the vertex of V_P on level $i + 1$. Similarly, for each i, $1 \leq i < k \cdot (k-1)/2$, the vertex of V_P on level $k + 2i$ has an arc to the vertex of V_P on level $k + 2i + 1$. We now show that the remaining vertices of V_P also have arcs to the vertices of V_P on the next level. Observe that these vertices and the vertices on the next level correspond to edges of H whose endpoints have the same unordered color pair. Hence, consider two vertices in V_P corresponding to two edges of H with the same unordered color pair $\{c(i), c(j)\}$. Since S is a clique, these two vertices in V_P are siblings, that is they correspond to the same undirected edge. Hence, there is in D an arc between the two vertices. Hence, $D[V_P]$ is a connected graph containing one vertex from each level, and thus, it is a path of length k^2.

$\Leftarrow$: Let P be a (D, G)-supported path of length k^2. Clearly, P contains exactly one vertex from each of the $1+k^2$ levels of D. Let $S := V(P) \cap W$. First, since P contains exactly one vertex from each level, S contains one vertex from each color class of H. It remains to show that every pair of vertices $u, v \in S$ is adjacent in H. Again, since P contains exactly one vertex from each level, it contains two vertices corresponding to the unordered color pair $\{c(u), c(v)\}$. Since there is an arc between these vertices, they are siblings, so we can call them $v_{i,j}$ and $v_{j,i}$, and we have $\{i, j\} \in F$. The only neighbors of these vertices in G are i and j, respectively. Since $G[V(P)]$ is connected, we thus have $\{i, j\} \in V(P)$. Furthermore, $c(i) \neq c(j)$, and thus, we can assume that $c(i) = c(u)$ and $c(j) = c(v)$. Since u is the only vertex in P with color $c(u)$, we have $i = u$. Similarly, $j = v$. Hence, u and v are adjacent in H. $\square$

Note that we can adapt the construction slightly in order to show W[1]-hardness for the problem of finding a (D, G)-supported path of length at most k. To achieve this, add two additional vertices t and w to the construction. Then, make t in D an outneighbor of all vertices in level k and an inneighbor of all vertices in level $k + 1$. Moreover, remove all arcs between level k and $k + 1$. Make vertex w an outneighbor of all vertices in the last level of D. Now, modify G as follows: instead of making s adjacent to all vertices of the Levels 1 through k, make w adjacent to these vertices. Then, make t adjacent to s and s adjacent to w.

With this construction, one can ensure that every (D, G)-supported path takes one vertex from each layer plus t and w. This can be seen as follows (recall that any (D, G)-supported path must contain at least two vertices). If the path P contains vertex t, then it must contain s and, therefore, also w, since only w is in G adjacent to any further vertices. Similarly, if the path contains s, then it must contain w. If the path contains more than one vertex from Layer 1 through k, then it must contain w and, thus, also t and s. If the path contains any vertex from any other layer, then it must contain at least one vertex from Layers 1 through k, which implies that it contains t. Thus, in all cases, the path contains s and w, which means that any (D, G)-supported path, if it exists, has length $k^2 + 2$. All other arguments remain the same.

Corollary 1. *The problem of deciding whether there is a (D, G)-supported path of length at most k is W[1]-hard, even if D is acyclic and G is a tree of diameter four.*

5. Enumeration-Based Algorithms

We now describe two algorithms for deciding whether there is a (D, G)-supported path of length exactly k that begins in a vertex u and ends in a vertex v. Both algorithms work for general graphs G and D. The first algorithm enumerates the paths of length k starting in u.

Theorem 6. *Let $\tilde{\Delta}$ denote the minimum of the maximum outdegree and the maximum indegree of G. Given two vertices $u, v \in V$, one can determine in $O(\tilde{\Delta}^k \cdot |E|)$ time, by an algorithm that we call SPSearch(u, v, k), whether there is a (D, G)-supported path of length k from u to v.*

Proof. We describe the algorithm for the case that $\tilde{\Delta}$ is the maximum outdegree of D. The running time bound for the case that $\tilde{\Delta}$ is the maximum indegree of D can be obtained by using the same algorithm on the instance that is obtained by reversing all arcs of D and searching for a (D, G)-supported path from v to u.

The algorithm *SPSearch*(u, v, k) is a search tree algorithm. At the root, set $P = (u)$, that is P is the trivial path of length zero that contains only u. At each search tree node, we check whether there is a (D, G)-supported path that has P as the prefix. If P has length k, P is a path from u to v and $G[V(P)]$ is connected, then return P. Otherwise, abort the branch. If the length of P is at most $k - 1$, then find a solution recursively by extending P. That is, for each arc (w, x) in D, where $x \notin V(P)$, search recursively for a solution whose prefix is $(u, \ldots, w, x)$ (the condition $x \notin V(P)$ is due to the fact that the solution has to be a simple path).

This approach clearly finds a (D, G)-supported path from u to v if it exists. The search tree has depth k, and in each search tree node, the number of new branches is at most $\tilde{\Delta}$. Moreover, in each search tree node, we can check in $O(|E|)$ time whether $G[V(P)]$ is connected. □

The second algorithm relies on an enumeration algorithm for connected subgraphs of order k in undirected graphs [25]. In the running time bound, e denotes Euler's number.

Theorem 7. *Let Δ be the maximum degree of G and $\tilde{\Delta}$ be the minimum of the maximum outdegree and maximum indegree of D. Given two vertices $u, v \in V$, one can determine in $O((2 \cdot e \cdot \Delta)^k \cdot (\Delta + k) \cdot \tilde{\Delta} k^3)$ time whether there is a (D, G)-supported path of length k from u to v.*

Proof. Assume first that a vertex set S of size k is given and that we need to determine whether $D[S]$ has a (D, G)-supported path from u to v of length $|S| - 1$. That is, we check whether $D[S]$ has a Hamiltonian path that starts in u and ends in v. Since $|S| = k$, this can be done in $O(2^k \cdot k^2 + \tilde{\Delta} k)$ time by first building $D[S]$ in $O(\tilde{\Delta} k)$ time and then applying the standard dynamic programming algorithm for the TRAVELING SALESMAN [26,27].

Now, to obtain S or to determine that no suitable S exists, we can use the fact that all connected subgraphs of G that contain u and have k vertices can be enumerated in $O((e \cdot (\Delta - 1))^k \cdot (\Delta + k)k)$ time [25]. The overall running time bound follows by multiplying the number of enumerated subgraphs by the time needed to verify the existence of a Hamiltonian path in each $D[S]$. □

6. Reduction Rules

We now present some data reduction rules that are used in our implementation. The first two reduction rules are obvious and can be applied in the general case when we do not search for a (D, G)-supported path between u and v, but for any shortest or longest (D, G)-supported path.

Rule 1. *If D contains an arc (a, b), such that a and b are in different connected components of G, then remove (a, b) from D.*

This rule is correct since the arc (a, b) cannot be part of any (D, G)-supported path.

Rule 2. *If G contains an edge $\{a, b\}$, such that there is no path from a to b and no path from b to a in D, then remove $\{a, b\}$ from G.*

This rule is correct, since a and b cannot be part of any (D, G)-supported path. The next rule is trivial; we mention it since it is part of our implementation.

Rule 3. *Remove all vertices that are isolated in either G or D.*

The next two rules apply to the problem when we search a (D, G)-supported path of length at most k from u to v. Their correctness is obvious as they delete vertices that cannot be in any (D, G)-supported path of length at most k from u to v.

Rule 4. *If V contains a vertex w, such that the sum of the length of the shortest path from u to w in D and the length of the shortest path w to v is more than k, then remove w from D and G.*

Rule 5. *If V contains a vertex w that is in G in a different component than u or v, then remove w from D and G. If $w = u$ or $w = v$, then return "no".*

Our final rule removes vertices that can be reached from u and that can reach v, but that need to visit the same vertex on both paths.

Rule 6. *If V contains a vertex w and a vertex x such that, in $D[V \setminus \{w\}]$, there is no path from u to x and no path from x to v, then remove x from D and G.*

The correctness of the rule follows from the fact that every path from u to x contains w, and every path from x to v also contains w. Hence, every path from u to v that visits x is not simple. As we will discuss in Section 7, these reduction rules considerably reduce the size and the maximum degrees of the biological instances.

7. Experiments

To assess the range of instances that can be solved by enumeration-based algorithms, we performed experiments on biological and synthetic data. We implemented and evaluated both algorithms.

Experimental Setup and Implementation Details. Our algorithms were implemented in Python, Version 2.7.3; source code and data are publicly available [28]. We used the NetworkX package for the graph data structures and basic graph algorithms, such as computing the shortest path. The test machine is a 4-core 3.6-GHz Intel Xeon E5-1620 (Sandy Bridge-E) with 10 MB L3 cache and 64 GB main memory, running under Debian GNU/Linux 7.0. We report wall-clock times. In all experiments, a timeout threshold of 30 seconds was used. The reported running times do not include the time for the first pass of Rules 1 to 3 (as it was performed only once in advance for the biological instance), but they include the time needed for the exhaustive application of all other data reduction rules. The first pass of Rules 1 to 3 takes less than 0.01 s in all instances and is thus negligible.

The algorithm that enumerates paths, which we will call the path enumeration algorithm, is implemented as follows. Given a pair of vertices u and v, it tries to determine whether there is a (D, G)-supported path from u to v as follows. First, perform the data reduction rules exhaustively. Then, determine for increasing values of k whether there is a (D, G)-supported path of length at most k from u to v. This is done by invoking the enumeration algorithm *SPSearch*(u, v, k) in Theorem 6. The initial k is the maximum of the length of a shortest path from u to v in D and in G.

Moreover, when branching into the cases to extend a path $(u, \ldots, w)$ by a vertex x, we perform a check based on the following observation.

Observation 1. *If $(u, \ldots, w)$ is a path of length ℓ and the shortest path from x to v has length at least $k - \ell$, then there is no (D, G)-supported path of length at most k with prefix $(u, \ldots, w, x)$.*

That is, we check for each arc (w, x) whether taking the shortest path from x to v gives a path of length at most k. If this is not the case, then x is discarded.

In preliminary experiments, we also tested whether reversing D and starting the search from v instead of u has an impact on the running time. Unfortunately, even if one runs both (direct and reverse) algorithms in parallel (that is, one only retains the minimum of the running times of the two algorithms), this approach does not yield a substantial speedup.

For the algorithm that enumerates connected subgraphs of G, that we call the tree enumeration algorithm (since it essentially enumerates spanning trees of subgraphs of G), we used a simple procedure for the subgraph enumeration that maintains a set S, such that $u \in S$ and $G[S]$ is connected. If $|S| = k+1$ and $v \in S$, then the algorithm checks whether $D[S]$ has a Hamiltonian path from u to v. Otherwise, it recursively extends S by considering each neighbor of a vertex in S. In this procedure, we use the two following observations for early termination of the subgraph enumeration. The first observation identifies sets that are too far from v.

Observation 2. *If $v \notin S$ and all vertices in S have distance at least $k - |S| + 2$ to v, then there is no order-$(k+1)$ set S' such that $S \subset S'$, $v \in S'$ and $G[S']$ is connected.*

The second observation identifies sets S such that the number of arcs in $D[S]$ is too low to obtain a directed path from u to v.

Observation 3. *If $D[S]$ has less than $k - (2(k + 1 - |S|))$ arcs, then there is no order-$(k+1)$ set S' such that $S \subset S'$, $D[S']$ has a Hamiltonian path.*

The correctness stems from the fact that the desired path has k arcs and that every deletion of a vertex can remove at most two path arcs from the graph. Unfortunately, the second observation yields a reduction only if $|S| > k/2$. Nevertheless, it proved to be crucial for this algorithm, as without the corresponding early termination rule, the number of enumerated subgraphs is too large.

Biological Data. We downloaded the biochemical pathways information from the *Saccharomyces* Genome Database (SGD) [29], which contains for each enzyme of *S. cerevisiae* a mapping from its EC (enzyme catalog) number to its gene identifier. Then, also from SGD, we obtained the protein interactions for the proteins encoded by these genes. To obtain the reactions from the EC numbers, we used the Explore Enz database [30]: For each pair of EC numbers x and y, we checked whether there is at least one product of x that is a substrate of y. If this were the case, then the arc (g_x, g_y) was added to D, where g_x and g_y are the gene identifiers associated with the EC numbers x and y, respectively. To restrict the product/substrate relationships in the directed graph to meaningful ones, we excluded five frequent metabolites ("ATP", "ADP", "AMP", "H_2O" and "H^+") that have high abundance in the cell. For the resulting pair of directed graph D and undirected graph G, we randomly sampled 500 vertex pairs and tried to determine for each pair (u, v) whether there is a (D, G)-supported path from u to v.

Synthetic Data. We performed experiments on random graphs with $n = 50$ and $n = 100$ vertices. The undirected graphs are created using the $G_{n,p}$-model due to Erdős, Rényi and Gilbert, where each edge is present with a fixed probability p. Similarly, in the directed graph, each arc is present with a fixed

probability p, that is for each pair of vertices u and v, the arc (u, v) is present with probability p and the arc (v, u) is also present with probability p. We use p_G to denote the probability used for generating G and p_D to denote the one for generating D.

For each value of n, we created two classes of instances: (i) sparse instances for which either $p_G = 0.05$ and $p_D = 0.1$ or $p_G = 0.1$ and $p_D = 0.05$; and (ii) dense instances for which either $p_G = 0.1$ and $p_D = 0.5$ or $p_G = 0.5$ and $p_D = 0.1$. For each triple of values of n, p_G and p_D, we created 500 instances and randomly picked a pair of vertices u and v such that we aim to find a (D, G)-supported path from u to v (we chose a higher number of pairs for the real-world instance to compensate for the fact that we considered only one biological instance).

Experimental Results. We first describe the experiments on the biological data. Some properties of the two networks and the effect of the data reduction rules are shown in Table 2. It can be seen that the data reduction substantially reduces the size of the remaining instances. In addition, for 358 of 500 random endpoint pairs, the data reduction solved the instance directly (by removing either v or u). Our running time results for both algorithms are shown in Figure 4. As expected, searching only for short paths is much easier than considering all solutions. In particular, we conclude that (D, G)-supported paths of length at most eight can be found quickly with both algorithms. Path enumeration is consistently faster, but the difference is slightly less pronounced for $k \leq 8$.

Table 2. Instance properties and effect of data reduction. Here, n denotes the number of vertices, m_D and m_G denote the number of edges in D and G, Δ^+ denotes the maximum outdegree in D, Δ^- denotes the maximum outdegree in D, Δ denotes the maximum degree in G and c_D and c_G denote the number of connected components in D and G. The values in the second line are arithmetic means over those instances that where not solved by the data reduction; Rule 4 is excluded since it depends on k.

	n	m_D	Δ^+	Δ^-	c_D	m_G	Δ	c_G
input	301	1221	35	49	6	1402	59	2
after Rules 1–3, 5, 6	104.6	526.9	25.8	24.4	1	270.4	22.2	1

Our results for the experiments with synthetic data are shown in Figures 5 and 6. Our main observations here are that sparser instances are harder than dense instances. For the path enumeration algorithm, the case when G is sparse compared to D is difficult. Here, the tree enumeration algorithm performs better both in sparse and in dense instances. The reason is most likely that in this case, short paths in D are unlikely to induce connected graphs in G. Conversely, the case in which G is dense compared to D appears to be easily solvable by the path enumeration algorithm on graphs with up to 100 vertices. Our experiments furthermore suggest that running times depend less on the vertex number than on the density of the respective graphs. Data reduction is much less effective on synthetic data: it very rarely reduces the number of edges or vertices at all. This suggests that the data reduction rules efficiently exploit the structure of real-world instances. Altogether, the experiments show that it may make sense to first compute the density of the two input graphs and then to choose the correct algorithm.

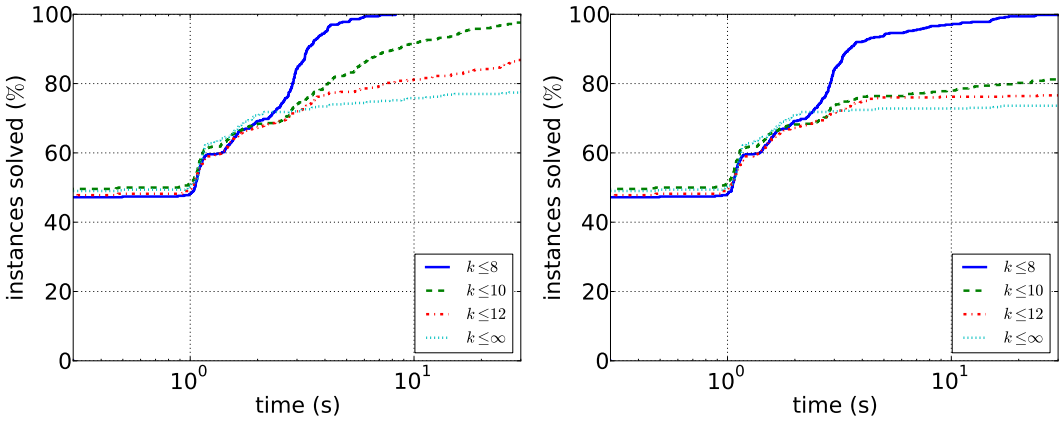

Figure 4. Running times for the biological data. (**Left**) The running times of the path enumeration algorithm with restricted and unrestricted path length; (**right**) the running times of the tree enumeration algorithm.

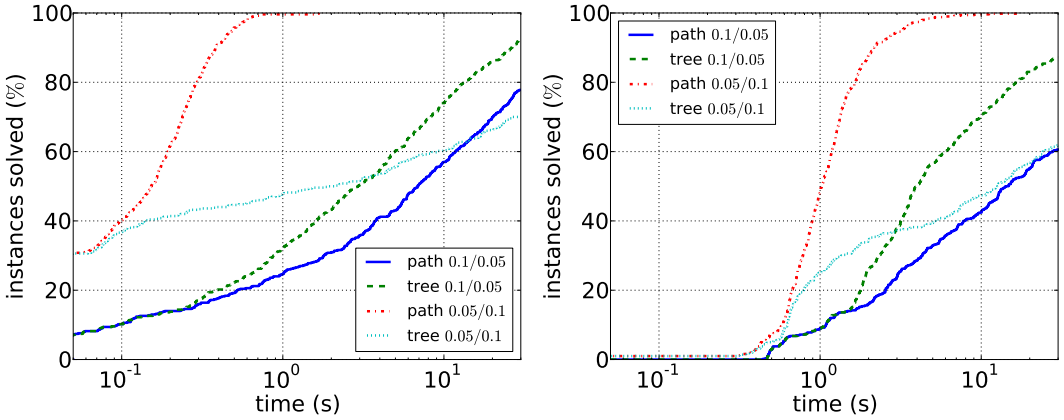

Figure 5. Comparison of the path enumeration algorithm and the tree enumeration algorithm in sparse synthetic data. In the legend, the first number is p_D and the second number is p_G. (**Left**) $n = 50$; (**right**) $n = 100$.

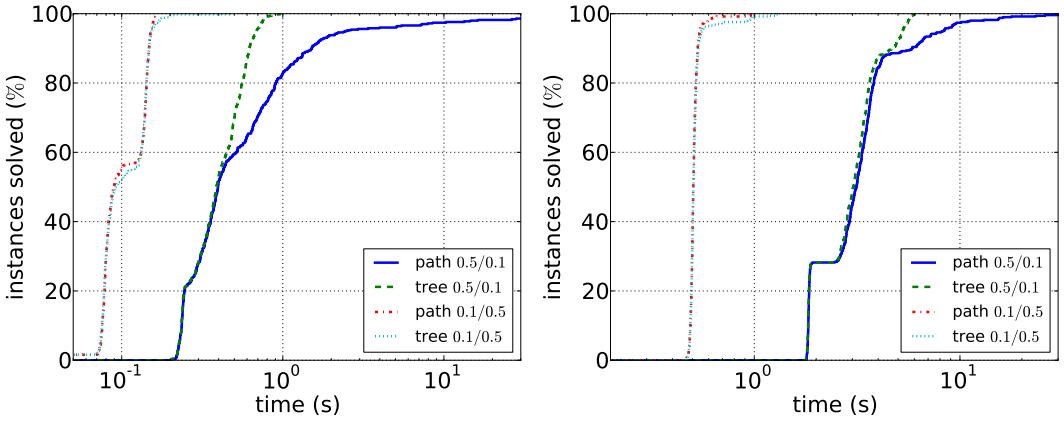

Figure 6. Comparison of the path enumeration algorithm and the tree enumeration algorithm in dense synthetic data. In the legend, the first number is p_D and the second number is p_G. (**Left**) $n = 50$; (**right**) $n = 100$.

As a final remark on random graphs, the fact that dense graphs are easily solvable is not surprising: for a fixed vertex w, the probability that there is a path (u, w, v) in D is $p_D \cdot p_D$. The probability that $G[\{u, v, w\}]$ is connected is: $p_G^3 + 3 \cdot p_G^2 \cdot (1 - p_G)$. Thus, for constant values of p_D and p_G, the probability that a vertex pair (u, v) has a (D, G)-supported path in n-vertex graph is one in the limit.

Biological Examples. A short (D, G)-supported path that was found in the biological dataset is ENO2 $\rightarrow$ ARO1 $\rightarrow$ BNA4 $\rightarrow$ ALD6. Interestingly, ENO2 and ALD6 encode enzymes of the glucose fermentation pathway, whereas the enzymes encoded by ARO1 and BNA4 are not annotated as parts of this pathway. Ideally, the existence of an alternative short (D, G)-supported path between ENO2 and ALD6 could either point to alternative (sub)pathways for glucose fermentation or to missing data in the yeast protein interaction network.

The longest (D, G)-supported path that was found is the length 13 path HOM6 $\rightarrow$ HOM2 $\rightarrow$ BNA4 $\rightarrow$ ARO1 $\rightarrow$ ARO2 $\rightarrow$ TRP3 $\rightarrow$ GDH3 $\rightarrow$ AAT2 $\rightarrow$ GDH2 $\rightarrow$ SER1 $\rightarrow$ ALT1 $\rightarrow$ ARO9 $\rightarrow$ AGX1 $\rightarrow$ GCV2. Statistical analysis using YeastMine [31] (where the test calculates hypergeometric enrichment p-values, the significance level was set to 0.05 and the p-values were corrected with the Holm–Bonferroni method) shows significant enrichment of annotation in more than 20 different GO (gene ontology) terms; all genes are annotated as parts of the "cellular amino acid metabolic process". There is only one pathway annotation with significant enrichment: the "superpathway of phenylalanine, tyrosine and tryptophan biosynthesis"; only four genes are annotated as parts of this pathway. This discrepancy might point to missing pathway annotations.

For such direct biological conclusions, however, a more extensive analysis is necessary. In particular, a more careful construction of the metabolic network might be needed. In addition, adding further side constraints may help in finding better biological solutions. Nevertheless, the general approach of including data from two networks may prove useful, for example in the discovery of new pathways.

8. Conclusions

SUPPORTED PATH belongs to a new type of subnetwork mining problems, arising from recent applications of biological, social or information networks: several graphs, of various types, represent different relations between objects, and a subset of objects is sought, with particular properties in each network. Our study of SUPPORTED PATH shows that these new problems can be very hard, even if one of the input networks has a very restricted structure. For the particular case of SUPPORTED PATH and its variants, we showed, for example, that the problem remains hard even if D is acyclic and G is a tree. We also provided several initial positive results, for example two algorithms for finding (D, G)-supported paths, that we tested on both biological and synthetic data. In particular, this allowed us to show that short (D, G)-supported paths can be found efficiently on realistic data.

Only two cases are left open by our complexity analysis: D is acyclic; D^* is outerplanar; and G is either a tree or a general graph. For these two input restrictions, the complexities of SUPPORTED PATH and of SHORTEST SUPPORTED PATH are open. It would be interesting to determine if these problems are polynomial or NP-complete.

Further studies could either investigate the complexity of SUPPORTED PATH in terms of approximability (on specific graph classes) or inexact variants of the problem obtained, for instance, by allowing small differences between the vertex set of the path in D and the connected set in G. It is also an open problem to find efficient exact exponential-time algorithms with a running time of $O(c^n)$ for some small c. As LONGEST SUPPORTED PATH is already a generalization of a DIRECTED HAMILTONIAN PATH, it might be interesting to focus either on SUPPORTED PATH and SHORTEST SUPPORTED PATH or on special cases of the problem, for example the case that D is acyclic. Furthermore, it is open to identify cases in which SUPPORTED PATH is polynomial-time solvable, but SHORTEST SUPPORTED PATH is NP-hard. Finally, our results show that finding supported paths remains hard if both D^* and G have constant treewidth. The graph (V, E') where E' is the union of the edges of D^* and G, however, might have unbounded treewidth. Hence, the complexity of all three problems remains open for the case that (V, E') has bounded treewidth.

Concerning the implementation, we showed that a simple approach can find (D, G)-supported paths of length at most eight. With a more efficient implementation of the graph data structures and shortest path algorithms, it should be possible to increase the "maximum tractable" path length by one or two. To find even longer paths, either further data reduction rules or more efficient early termination rules could be promising approaches. Finally, it could be interesting to study heuristic approaches that reduce the general problems that are very hard to restricted variants; for example, to the variant in which D is acyclic [32].

Acknowledgments

The authors would like to thank the referees for their detailed and constructive remarks that substantially helped to improve the paper.

Christian Komusiewicz was partially supported by a post-doctoral grant funded by the Région Pays de la Loire and by the DAAD Procope program (Project 55934856).

Author Contributions

The problem definition, theoretical analysis, and algorithm design were performed jointly by all authors; C.K. implemented the algorithms and performed the experiments.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Cai, D.; Shao, Z.; He, X.; Yan, X.; Han, J. Mining Hidden Community in Heterogeneous Social Networks. In Proceedings of the ACM-SIGKDD Workshop on Link Discovery: Issues, Approaches and Applications (LinkKDD 2005), Chicago, IL, USA, 21–25 August 2005; pp. 58–65.

2. Matsuo, Y.; Hamasaki, M.; Takeda, H.; Mori, J.; Bollegara, D.; Nakamura, Y.; Nishimura, T.; Hasida, K.; Ishizuka, M. Spinning Multiple Social Networks for Semantic Web. In Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI 2006), Boston, MA, USA, 16–20 July 2006; pp. 1381–1387.
3. Vicentini, R.; Menossi, M. *Data Mining and Knowledge Discovery in Real Life Applications*; Ponce, J., Karahoca, A., Eds.; In-Tech: Croatia, Rijeka, 2009.
4. Bunke, H. Graph matching: Theoretical foundations, algorithms and applications. In Proceedings of the International Conference on Vision Interface (VI 2000), Montréal, QC, Canada, 14–17 May 2000; pp. 82–88.
5. Conte, D.; Foggia, P.; Sansone, C.; Vento, M. Thirty years of graph matching in pattern recognition. *Int. J. Pattern Recogn. Artif. Intell.* **2004**, *18*, 265–298.
6. Džeroski, S. *Relational Data Mining*; Springer: Berlin, Germany, 2010.
7. Kelley, B.P.; Yuan, B.; Lewitter, F.; Sharan, R.; Stockwell, B.R.; Ideker, T. PathBLAST: A tool for alignment of protein interaction networks. *Nucleic Acids Res.* **2004**, *32*, doi:10.1093/nar/gkh411.
8. Sharan, R.; Suthram, S.; Kelley, R.M.; Kuhn, T.; Mccuine, S.; Uetz, P.; Sittler, T.; Karp, R.M.; Ideker, T. Conserved patterns of protein interaction in multiple species. *Proc. Natl. Acad. Sci. USA* **2005**, *102*, 1974–1979.
9. Wernicke, S.; Rasche, F. Simple and Fast Alignment of Metabolic Pathways by Exploiting Local Diversity. *Bioinformatics* **2007**, *23*, 1978–1985.
10. Pinter, R.Y.; Rokhlenko, O.; Yeger-Lotem, E.; Ziv-Ukelson, M. Alignment of metabolic pathways. *Bioinformatics* **2005**, *21*, 3401–3408.
11. Bourqui, R.; Lacroix, V.; Cottret, L.; Auber, D.; Mary, P.; Sagot, M.F.; Jourdan, F. Metabolic network visualization eliminating node redundance and preserving metabolic pathways. *BMC Syst. Biol.* **2007**, *1*, doi:10.1186/1752-0509-1-29.
12. Planes, F.; Beasley, J. A critical examination of stoichiometric and path-finding approaches to metabolic pathways. *Brief. Bioinform.* **2008**, *9*, 422–436.
13. Zubarev, R.; Nielsen, M.; Fung, E.; Savitski, M.; Kel-Margoulis, O.; Wingender, E.; Kel, A. Identification of dominant signaling pathways from proteomics expression data. *J. Proteom.* **2008**, *71*, 89–96.
14. Bruckner, S.; Hüffner, F.; Karp, R.M.; Shamir, R.; Sharan, R. Topology-Free Querying of Protein Interaction Networks. *J. Comput. Biol.* **2010**, *17*, 237–252.
15. Boyer, F.; Morgat, A.; Labarre, L.; Pothier, J.; Viari, A. Syntons, metabolons and interactons: An exact graph-theoretical approach for exploring neighbourhood between genomic and functional data. *Bioinformatics* **2005**, *21*, 4209–4215.
16. Durek, P.; Walther, D. The integrated analysis of metabolic and protein interaction networks reveals novel molecular organizing principles. *BMC Syst. Biol.* **2008**, *2*, doi:10.1186/1752-0509-2-100.
17. Williams, E.; Bowles, D.J. Coexpression of neighboring genes in the genome of Arabidopsis thaliana. *Genome Res.* **2004**, *14*, 1060–1067.
18. Downey, R.G.; Fellows, M.R. *Parameterized Complexity*; Springer: Berlin, Germany, 1999.

19. Flum, J.; Grohe, M. *Parameterized Complexity Theory*; Springer: Berlin, Germany, 2006.
20. Niedermeier, R. *Invitation to Fixed-Parameter Algorithms*; Oxford University Press: Oxford, UK, 2006.
21. Hartung, S.; Nichterlein, A. Technische Universität Berlin. Personal communication, 2013.
22. Fortune, S.; Hopcroft, J.; Wyllie, J. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.* **1980**, *10*, 111–121.
23. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*; W. H. Freeman and Company: San Francisco, CA, USA, 1979.
24. Fellows, M.R.; Hermelin, D.; Rosamond, F.A.; Vialette, S. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.* **2009**, *410*, 53–61.
25. Komusiewicz, C.; Sorge, M. An Algorithmic Framework for Fixed-Cardinality Optimization in Sparse Graphs Applied to Dense Subgraph Problems. *Discret. Appl. Math.* **2015**, in press.
26. Bellman, R. Dynamic Programming Treatment of the Traveling Salesman Problem. *J. ACM* **1962**, *9*, 61–63.
27. Held, M.; Karp, R.M. A dynamic programming approach to sequencing problems. *J. Soc. Ind. Appl. Math.* **1962**, *10*, 196–210.
28. Supported Path Software. Available online: http://fpt.akt.tu-berlin.de/supported-path (accessed on 8 October 2015).
29. Cherry, J.M.; Hong, E.L.; Amundsen, C.; Balakrishnan, R.; Binkley, G.; Chan, E.T.; Christie, K.R.; Costanzo, M.C.; Dwight, S.S.; Engel, S.R.; *et al*. Saccharomyces Genome Database: The genomics resource of budding yeast. *Nucleic Acids Res.* **2012**, *40*, doi:10.1093/nar/gkr1029.
30. McDonald, A.; Boyce, S.; Moss, G.; Dixon, H.; Tipton, K. ExplorEnz: A MySQL database of the IUBMB enzyme nomenclature. *BMC Biochem.* **2007**, *8*, doi:10.1186/1471-2091-8-14.
31. Balakrishnan, R.; Park, J.; Karra, K.; Hitz, B.C.; Binkley, G.; Hong, E.L.; Sullivan, J.; Micklem, G.; Cherry, J.M. YeastMine—An integrated data warehouse for Saccharomyces cerevisiae data as a multipurpose tool-kit. *Database* **2012**, doi:10.1093/database/bar062.
32. Blin, G.; Fertin, G.; Mohamed-Babou, H.; Rusu, I.; Sikora, F.; Vialette, S. Algorithmic Aspects of Heterogeneous Biological Networks Comparison. In Proceedings of the 5th International Conference on Combinatorial Optimization and Applications (COCOA 2011), Zhangjiajie, China, 4–6 August 2011; pp. 272–286.