# Growing DAGs: Optimization Functions for Pathway Reconstruction Algorithms

Tunç Başar Köse[1,†], Jiarong Li[1,2,†], and Anna Ritz[3,*]

[1]Computer Science Department, Reed College, Portland, OR, USA
[2]Current Affiliation: Microsoft Cooperation
[3]Biology Department, Reed College, Portland, OR, USA
[†]These Authors Contributed Equally.
[*]aritz@reed.edu

**Abstract**

A major challenge in molecular systems biology is to understand how proteins work to transmit external signals to changes in gene expression. Computationally reconstructing these signaling pathways from protein interaction networks can help understand what is missing from existing pathway databases. We formulate a new pathway reconstruction problem, one that iteratively grows directed acyclic graphs (DAGs) from a set of starting proteins in a protein interaction network. We present an algorithm that provably returns the optimal DAGs for two different cost functions, and show how the pathway reconstructions have different topological features for each cost function when applied to six diverse signaling pathways from the NetPath database. Growing DAGs is a promising step towards reconstructing pathways with specific network topologies.

## 1 Motivation

Intracellular signaling pathways describe the molecules and interactions that convert a particular external signal (such as growth, proliferation, movement, or death) to the change of expression of one or more genes, culminating in a cellular response through transcriptional regulation. Many signaling pathway databases such as Reactome [1], KEGG [2], and NetPath [3] document the interactions associated with signaling pathways, and efforts such as WikiPathways [4] and Pathway Commons [5] have unified these databases by combining dozens of pathway resources. However, even pathways that describe fundamental biological processes or pathways implicated in commonly studied diseases are not complete: they are likely missing canonical proteins and protein interactions. Further, it is difficult to use these pathway databases to study less well-known signaling events. Computationally reconstructing a signaling pathway of interest from experimental data would be a huge help for the (mostly manual) curation of pathway databases such as Reactome and KEGG, and would provide a new lens to investigate signaling pathways that are not yet cataloged in these databases.

We and others have made use of large protein-protein interaction datasets as well as the annotated pathway databases to construct an *interactome* – a graph representation of physical interactions among all proteins in the organism under study. Since some interactions (especially those from existing databases) represent post-translational modifications where a direction of signal is clear, we consider an interactome as a weighted, directed graph $G = (V, E)$ where the edges are weighted according to the supporting evidence.

### 1.1 Pathway Reconstruction Problem

In earlier work [7], we formulated the following **Pathway Reconstruction Problem**: Given a weighted, directed interactome $G = (V, E)$, a set $R \subset V$ of membrane-bound receptors specific to a pathway of interest, and a set $T \subset V$ of transcriptional regulators specific to a pathway of interest, return a subgraph $G' \subseteq G$ that corresponds to the signaling pathway that connects nodes in $R$ to nodes in $T$. We showed that a $k$-shortest paths (KSP) approach, PathLinker, outperformed existing methods for connecting nodes within a network to reconstruct pathways in pathway databases. There are two main reasons why a KSP approach such as PathLinker reconstructed pathways better that other methods. First, A KSP approach has a parameter to smoothly "grow" the network. Many existing algorithms return a small reconstruction because they aim to minimize the number of extraneous edges (such as Steiner forests [8] and network flow [11]). In KSP, we can increase the number of paths $k$ to return iteratively larger reconstructions, capturing more of the pathway. Second, A KSP approach is guaranteed to connect nodes in $R$ to nodes in $T$. While some methods such as Random Walks with Restarts (RWR) [10] have a parameter that "grows" the network, they are not guaranteed to connect the nodes from $R$ to $T$. In a KSP approach, all paths start and end within the pathway by construction. Finally, reconstructions that are comprised of shortest paths are useful for generating hypotheses for follow-up validation. For example, we experimentally validated a path (RYK-CFTR-DAB2) from PathLinker's reconstruction of the Wnt signaling pathway and showed that CFTR is associated with Wnt signaling through interactions between RYK, a non-canonical receptor, and DAB2, an inhibitor of canconical Wnt signaling [7].

Despite PathLinker's success, calculating the first 2,000 shortest paths from receptors to transcriptional regulators for each signaling pathway in the NetPath databases captured only about 70% of the known signaling interactions [7], leaving much room for improvement. Extensions of PathLinker have used auxiliary data such as protein localization information [12] or additional downstream processing [13] to accurately reconstruct ground truth pathways. Other extensions have constrained the paths from $R$ to $T$ to follow regular expression patterns [14].

### 1.2 Contributions

Inspired by the success of KSP approaches, we reframe the Pathway Reconstruction Problem to directly optimize an objective function related to path cost on directed acyclic graphs (DAGs). We acknowledge up front that cycles (in the form of feedback loops) are important in signaling pathways, so this problem formulation is narrower in scope than the original Pathway Reconstruction Problem. Next, we present a collection of pathway reconstruction methods that solve this new variant of the problem. We analyze the topologies of pathway reconstruction methods with two different objective functions on a diverse set of six signaling pathways. Finally, we highlight our method's ability to reconstruct pathways starting from a larger seed DAG, such as a DAG constructed from known pathway interactions.

## 2 Methods

Let the interactome $G = (V, E)$ be directed with edge weights $w_{uv}$ for every edge $(u, v) \in E$. Given $R, T \subset V$ denoting the set of receptors and set of transcription factors for a particular pathway of interest, we first modify the graph by (a) introducing a super source node $s$ to $G$, adding directed edges with 0 edge weight from $s$ to each node in $R$, and removing all other incoming edges to nodes in $R$ and (b) introducing a super sink node $t$ to $G$, adding directed edges with 0 edge weight from each node in $T$ to $t$, and removing all other outgoing edges from nodes in $T$. We now focus on finding

$s$-$t$ paths in $G$, which corresponds to finding paths from *any* receptor in $R$ to *any* transcriptional regulator in $T$ that have the same cost as the original path lengths from receptors to transcriptional regulators. Further, nodes in $R$ and $T$ will only start and end the paths (they will never be internal nodes on paths). KSP approaches are parameterized by $k$, the number of shortest paths from $s$ to $t$. KSP approaches iteratively "grow" a subgraph $G_k$ of $G$ by taking the union of the first $k$ shortest paths from $s$ to $t$: $G_1, G_2, \ldots, G_j, \ldots, G_k$. In this way, KSP approaches iteratively "grow" a subgraph of $G$. Our goal is to grow directed acyclic graphs (DAGs) from a graph $G$.

### 2.1 A Problem Formulation for Growing DAGs

Let $G_0 \subset G$ be any DAG that connects $s$ and $t$ (for example, the shortest $s$-$t$ path). We will also keep track of all paths in a DAG: let $P_j(s,t)$ be the set of paths from $s$ to $t$ in DAG $G_j$. Finally, we define the new edges added to $G_j$ to be $E_\Delta = E_j \setminus E_{j-1}$ and the set of new paths added to be $P_\Delta = P_j(s,t) \setminus P_{j-1}(s,t)$.

**The Growing DAG Problem.** Given a weighted, undirected graph $G = (V, E, w)$ modified with super-source and super-sink nodes $s$ and $t$, a DAG $G_0 \subset G$ that connects $s$ and $t$, and a parameter $k$. For $j = 1, 2, \ldots, k$, find a DAG $G_j = (V_j, E_j, w_j)$ where $G_{j-1} \subset G_j \subseteq G$ such that

1. $P_\Delta \neq \emptyset$ (there exists at least one new $s$-$t$ path).
2. $\bigcup_{P_\Delta} = E_\Delta$ (all new edges in $G_j$ are on some $s$-$t$ path).
3. $G_j$ minimizes some cost function $c : G_j \mapsto \mathbb{R}$, which may be one of the following:

   **min_edge_cost:** $c_1(G_j) = \sum_{(u,v)\in E_j} w_{uv}$.
   **min_paths_cost:** $c_2(G_j, s, t) = \sum_{p\in P_j(s,t)} \sum_{(u,v)\in p} w_{uv}$.

Cost function **min_edge_cost** is simply the cost of the edges in the subgraph $G_j$, while cost function **min_paths_cost** computes the cost of all paths in $G_j$. An example iteration is shown in Figure 1; note that the $G_1$ that minimizes the two costs functions are quite different in (C) and (D), though they both add at least one new path and all new edges are on some $s$-$t$ path.

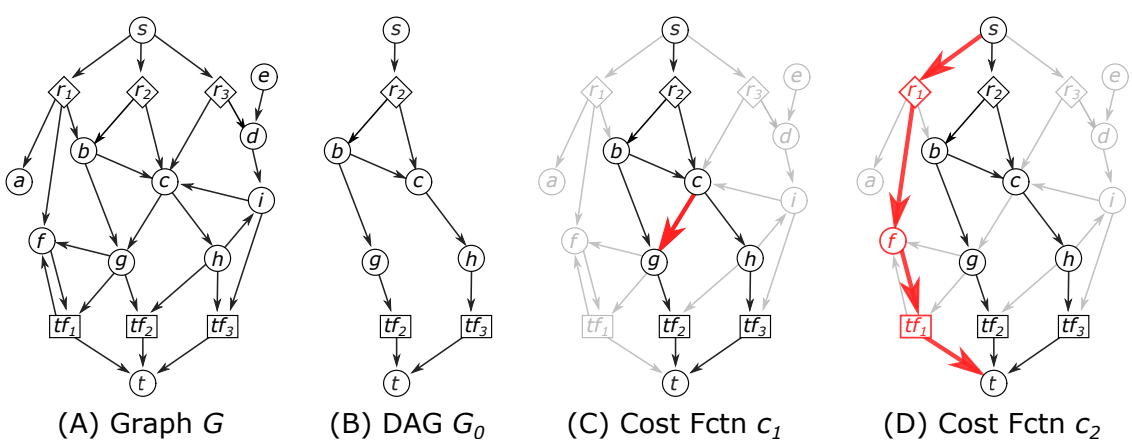


Figure 1: The Growing DAG Problem. (A) Input graph $G$ which may contain cycles and bidirected edges. In this example, all edges have unit cost. (B) An example DAG $G_0 \subset G$. (C) An example $G_1$ that minimizes **min_edge_cost** by adding a single edge to $G_0$. (D) An example $G_1$ that minimizes **min_paths_cost** by adding a path of length 4 to $G_0$.

**Algorithm 1** `PathCounter`$(G = (V, E), s, t)$

1: `nodes` $\leftarrow$ topologically sort $V$
2: `upstream` $\leftarrow \{\}$
3: **for** $u$ in `reverse(nodes)` **do**
4: $\quad$ `upstream[u]` $\leftarrow \begin{cases} \sum_{(u,v)\in E} \texttt{upstream[v]} & \text{if there exists some outgoing edge from } u \\ 1 & \text{if } u = t \\ 0 & \text{otherwise} \end{cases}$
5: **end for**
6: `downstream` $\leftarrow \{\}$
7: **for** $v$ in `nodes` **do**
8: $\quad$ `downstream[v]` $\leftarrow \begin{cases} \sum_{(u,v)\in E} \texttt{downstream[u]} & \text{if there exists some incoming edge to } v \\ 1 & \text{if } v = s \\ 0 & \text{otherwise} \end{cases}$
9: **end for**
10: **for** $(u, v) \in E$ **do**
11: $\quad f_{uv} =$ `upstream`$[v] \times$ `downstream`$[u]$
12: **end for**
13: **return** $f$

## 2.2 Calculating Cost Functions

In order to implement the Growing DAG Problem, we must show that we can efficiently determine if a DAG $G_j$ satisfies the properties listed and that we can efficiently identify such a DAG from the previous iteration. We first show that, given a DAG $G_j$, we can efficiently calculate the cost functions. Calculating **min_edge_cost** is straightforward: simply sum the edge weights in $G_j$. However, calculating **min_paths_cost** requires an enumeration of all paths in $G_j$. To calculate **min_paths_cost**, we want to compute the cost of all $s$-$t$ paths $P_j(s, t)$ in $G_j$ without having to enumerate all paths. It can be rewritten to calculate, for each edge in $E_j$, the number of $s$-$t$ paths that contain that edge. Let $f_{uv}$ be the number of paths in $P_j(s, t)$ that contain edge $(u, v)$:

$$c_2(G_j, s, t) = \sum_{p \in P_j(s,t)} \sum_{(u,v)\in p} w_{uv} = \sum_{(u,v)\in E_j} f_{uv} w_{uv}. \tag{1}$$

While counting the number of $s$-$t$ paths in a general graph is $\#P$-complete [16], we can efficiently count the number of $s$-$t$ paths in a DAG. Further, the dynamic program to count the number of $s$-$t$ paths in a DAG will also compute the number of $s$-$t$ paths $f_{uv}$ that pass through every edge $(u, v) \in E_j$ (Algorithm 1 and Figure 2). For every node $v \in G_j$, we can decompose the $s$-$t$ paths that go through $v$ into paths from $v$ to $t$ (Lines 2–5) and from $s$ to $v$ (Lines 6–9). In order to compute $f_{uv}$, we must calculate the number of $s$-$t$ paths that go through some edge $(u, v)$. This value can be calculated by multiplying `upstream[v]` and `downstream[u]` ((Lines 10–12)). Once we have $f$ from Algorithm 1, calculating **min_paths_cost** is straightforward using Equation (1).

## 2.3 Properties of an Optimal $G_j$

Now that we have shown that we can efficiently calculate the cost functions given a DAG $G_j$, we will describe how to identify the possible extensions of $G_{j-1}$ that guarantee that at least one new $s$-$t$ path is added to $G_j$. There may be multiple edges that are added to grow $G_j$ from $G_{j-1}$; let $G_\Delta = (V_j \setminus V_{j-1}, E_j \setminus E_{j-1})$ be the difference of $G_j$ and $G_{j-1}$. Since $G_{j-1}$ and $G_j$ are DAGs, the graph $G_\Delta$ will also be a DAG (which may be disconnected). We first prove some properties of $G_\Delta$ that hold for either of the cost functions listed.

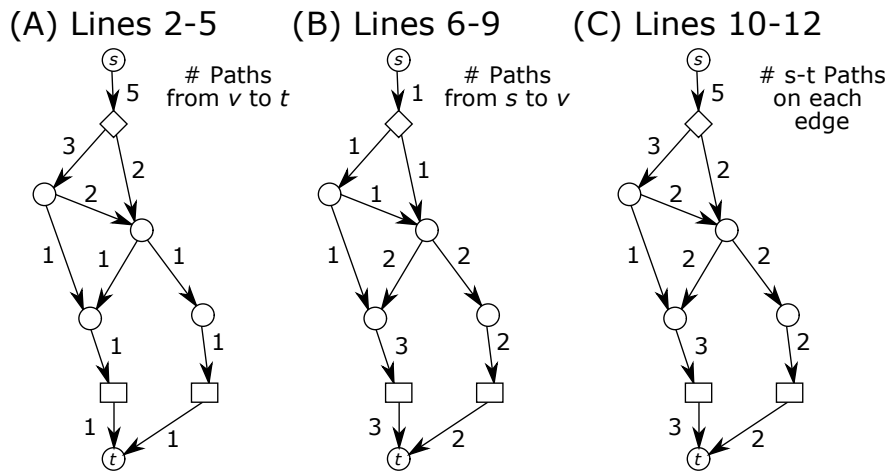


Figure 2: Illustration counting the total number of $s$-$t$ paths by (A) computing the number of paths from $v$ to $t$, (B) computing the number of paths from $s$ to $v$, and (C) combining these values to get the number of times each edge is traversed to count the total number of $s$-$t$ paths.

**Lemma 1.** *Given $G_{j-1}$ and $G_j$ that satisfy the Growing DAG problem, let $G_\Delta = G_j \setminus G_{j-1}$. The $G_\Delta$ that minimizes any of the cost functions will be exactly one path.*

*Proof.* $G_\Delta$ is a non-empty DAG, since at least one new $s$-$t$ path must exist in $G_j$. If $G_\Delta$ is not exactly one path, then $G_\Delta$ must be composed of multiple paths. Since all new edges in $G_j$ must be on some $s$-$t$ path, then every maximal path in $G_\Delta$ must start and end on some node from $G_{j-1}$ (which may include $s$ and $t$). Let $P = \{p_1, p_2, \ldots\}$ be the set of distinct maximal paths from $G_\Delta$.

If one path $p_i \in P$ establishes a new $s$-$t$ path in $G_j$, then other paths are unnecessary. Let $p_i$ start at some node $v_0$ and end at some node $v_k$; $p_i$ establishes a new $s$-$t$ path if $(s \rightsquigarrow v_0 \rightsquigarrow v_k \rightsquigarrow t)$ is in $G_j$. The other paths in $P$ will add extraneous edges that are not necessary to establish a new path, which increases cost function **min_edge_cost**. Further, the other paths in $P$ will add additional $s$-$t$ paths in $G_j$, which increases the cost function **min_paths_cost**. Since maximal paths can be dropped from $P$ to minimize either cost function, $G_\Delta$ is not optimal.

Suppose instead that multiple paths are needed to establish a new $s$-$t$ path in $G_j$. Without loss of generality, let two paths from $P$ be $v_0 \rightsquigarrow v_k$ and $u_0 \rightsquigarrow u_l$, and let a new $s$-$t$ path in $G_j$ be

$$s \rightsquigarrow v_0 \rightsquigarrow v_k \rightsquigarrow x \rightsquigarrow x' \rightsquigarrow u_0 \rightsquigarrow u_l \rightsquigarrow t,$$

where $x \rightsquigarrow x'$ is a path in $G_{j-1}$. There must be at least one edge from $G_{j-1}$ between $v_k$ and $u_0$ because otherwise $p_i$ and $p_j$ would form a single maximal path in $G_\Delta$. Since $G_{j-1}$ is a connected DAG, then $x \rightsquigarrow x'$ must be upstream of $t$ and/or downstream of $s$ in a topological ordering of $V_{j-1}$. Therefore, at least one of the following paths must exist:

1. $s \rightsquigarrow v_0 \rightsquigarrow v_k \rightsquigarrow x \rightsquigarrow x' \rightsquigarrow t$ if $x$ is upstream of $t$. In this case, $p_i$ is used but $p_j$ is extraneous and only increases the cost functions (by adding extra edges for **min_edge_cost** and adding extra $s$-$t$ paths for **min_paths_cost**), so $G_\Delta$ is not optimal.
2. $s \rightsquigarrow x \rightsquigarrow x' \rightsquigarrow u_0 \rightsquigarrow v_l \rightsquigarrow t$ if $x$ is downstream of $s$. In this case, $p_j$ is used but $p_i$ is extraneous and only increases the cost functions (by adding extra edges for **min_edge_cost** and adding extra $s$-$t$ paths for **min_paths_cost**), so $G_\Delta$ is not optimal.

□

**Lemma 2.** *$G_\Delta$ that minimizes the cost functions defines a path that starts at some node in $G_{j-1}$ and ends at some node in $G_{j-1}$. All internal nodes on the path are in $G_j$ but not in $G_{j-1}$.*

*Proof.* $G_\Delta$ is a single path $p = (v_0, v_1, \ldots, v_k)$ by Lemma 1. We need to show that $v_0$ and $v_k$ are in $V_{j-1}$, and all other nodes are not in $V_{j-1}$. First observe that there must be at least two nodes in $G_\Delta$ that are in $G_{j-1}$ for an $s$-$t$ path to include new edges in $G_j$ (which may include $s$ and/or $t$); call these nodes $x$ and $y$. Consider the path in $G_j$ $s \rightsquigarrow x \rightsquigarrow y \rightsquigarrow t$ where $x \rightsquigarrow y$ is a path in $G_\Delta$.

1. There are exactly two nodes $x, y \in V_{j-1}$ in the path $p = (v_0, v_1, \ldots, v_k)$. Suppose there was a third node, $z \in V_{j-1}$, in the path such that $x \rightsquigarrow z \rightsquigarrow y$. Since $G_{j-1}$ is a connected DAG, then $z$ must be upstream of $t$ and/or downstream of $s$. Therefore, at least one of the following paths must exist:
   (a) $s \rightsquigarrow x \rightsquigarrow z \rightsquigarrow t$ if $z$ is upstream of $t$. In this case, $z \rightsquigarrow y$ is extraneous and increases the cost functions (by adding extra edges for **min_edge_cost** and adding extra $s$-$t$ paths for **min_paths_cost**), so $G_\Delta$ is not optimal.
   (b) $s \rightsquigarrow z \rightsquigarrow y \rightsquigarrow t$ if $z$ is downstream of $s$. In this case, $x \rightsquigarrow z$ is extraneous and increases the cost functions (by adding extra edges for **min_edge_cost** and adding extra $s$-$t$ paths for **min_paths_cost**), so $G_\Delta$ is not optimal.
2. $x = v_0$. Suppose $x$ is some node in the path $v_i$ that is not $v_0$; the path from $(v_0, \ldots, v_{i-1})$ could be dropped, because $s \rightsquigarrow x$ will bypass these nodes.
3. $y = v_k$. Same argument as for $x = v_0$.

□

Together, Lemmas 1 and 2 prove the following theorem:

**Theorem 1.** *Given $G_{j-1}$ and $G_j$ that satisfy the Growing DAG problem, $G_{j-1}$ and $G_j$ differ by exactly one path that starts and ends in $G_{j-1}$ and contains no other nodes from $G_{j-1}$.*

### 2.4 The Growing DAG Algorithm

At each iteration $j$, we keep track of candidate paths for DAG $G_j$ using a modified Dijkstra's algorithm. We rely on topologically sorting the nodes in a DAG, which results in a partial ordering of the nodes. Let $\sigma(G)$ denote the partial ordering of a DAG $G$, with the position of each node $v$ denoted by $\sigma_v$. For a node $v$ in a DAG, its *ancestors* are all nodes $u$ where $\sigma_u < \sigma_v$ and its *descendants* are all nodes $x$ where $\sigma_v < \sigma_x$). Note that there may also be nodes $y$ that is neither an ancestor nor a descendant of $v$ (e.g., where $\sigma_v = \sigma_y$); we say that $v$ and $y$ are *incomparable*.

Algorithm 2 takes as input a directed, weighted graph $G$, an initial DAG $G_0$, the number of iterations $k$, and a cost function $c$ (either **min_edge_cost** or **min_paths_cost**). It returns a list of length $k$ that denotes the min-cost paths for each iterations according to the specified cost function. To track the set of candidate paths, we use a `dist` dictionary that stores the cost of the best paths between topologically sorted nodes in the DAG. We assume that we also have the predecessors so we can determine the path $u \rightsquigarrow v$ from `dist`$[u][v]$ as well.

Lines 3–8 initialize the `dist` dictionary for the nodes in $G_0$. First, we build a $G_{\text{cand}}$ graph that removes the existing DAG from $G$. Then, for every node $u$ in $G_0$ we consider, we must also remove any node that is an ancestor of $u$ to prevent paths that would induce cycles. Once we have the graph, we call a slightly modified Dijkstra's algorithm called `MultiTargetDijkstra`($G$,$s$,$T$) which takes three arguments: a graph $G$, a source node $s \in V$, and a set of target nodes $T \subset V$. `MultiTargetDijkstra`() uses Dijkstra's algorithm to find the shortest path from $s$ to each node in $T$, ensuring that a node in $T$ is never considered an internal node on a path from $s$ and returning

early if all nodes in $T$ are reached. The target nodes for $u$ are all of $u$'s descendants and incomparable nodes in $\sigma(G_0)$.

Once the distances are initialized, we iteratively grow $G_j$ for $j$ from 1 to $k$. First, we identify the min-cost path from `dist` according to cost function $c$ (Lines 10–21). This is simple for **min_edge_cost**, when we select the path $u \rightsquigarrow v$ with the lowest cost in `dist`. For **min_paths_cost**, we need to calculate the cost of all $s$-$t$ paths for every possible choice of valid $u \rightsquigarrow v$ path in `dist`. We can call Algorithm 1 on a graph $G'$ that contains the path we are checking in order to calculate the all-paths-cost $C[u][v]$ (Lines 16–18). We select the path $u \rightsquigarrow v$ with the lowest all-paths-cost $C[u][v]$, add the selected path to $G_{j-1}$ to make the $G_j$ DAG, and update the distances dictionary.

**Algorithm 2** `GrowDAGs`($G = (V, E)$,$G_0 = (V_0, E_0)$, num iters $k$, cost function $c$)

1: `paths` $\leftarrow$ []
2: `dist` $\leftarrow$ {}
3: *# Initialize distances*
4: $G_{\text{cand}} \leftarrow G \setminus G_0$
5: **for** $u \in V_0$ **do**
6: $\quad G_{\text{cand}}^{(u)} \leftarrow$ Remove any node $v$ from $G_{\text{cand}}$ where $\sigma_u > \sigma_v$ in $\sigma(G_0)$
7: $\quad$ `dist`$[u]$ = `MultiTargetDijkstra`($G_{\text{cand}}^{(u)}$,$u$,$\{v : \sigma_u \leq \sigma_v\}$ in $\sigma(G_0)$)
8: **end for**
9: **for** $j = 1, 2, \ldots, k$ **do**
10: $\quad$ *# Get the min cost path according to cost function c*
11: $\quad$ **if** $c = c_1$ **then**
12: $\quad\quad$ `paths`$[j] \leftarrow$ the path $u \rightsquigarrow v$ from `dist`$[u][v]$ such that $\min_{x,y}$ `dist`$[x][y]$ =`dist`$[u][v]$
13: $\quad$ **else if** $c = c_2$ **then**
14: $\quad\quad C \leftarrow \{\}$
15: $\quad\quad$ **for** $u, v$ such that $\sigma_u \leq \sigma_v$ in $\sigma(G_j)$ **do**
16: $\quad\quad\quad G' \leftarrow$ Add the path $u \rightsquigarrow v$ from `dist`$[u][v]$ to $G_{j-1}$
17: $\quad\quad\quad f \leftarrow$ `PathCounter`($G'$,$s$,$t$)
18: $\quad\quad\quad C[u][v] = \sum_{(u,v)\in E'} f_{uv} w_{uv}$
19: $\quad\quad$ **end for**
20: $\quad\quad$ `paths`$[j] \leftarrow$ the path $u \rightsquigarrow v$ from `dist`$[u][v]$ such that $\min_{x,y} C[x][y] = C[u][v]$
21: $\quad$ **end if**
22: $\quad$ *# Make $G_j$ and update distances*
23: $\quad G_j \leftarrow$ Add `paths`$[j]$ to $G_{j-1}$
24: $\quad G_{\text{cand}} \leftarrow G \setminus G_j$
25: $\quad$ **for** $u \in V_j$ **do**
26: $\quad\quad G_{\text{cand}}^{(u)} \leftarrow$ Remove any node $v$ from $G_{\text{cand}}$ where $\sigma_u > \sigma_v$ in $\sigma(G_j)$
27: $\quad\quad$ `dist`$[u]$ = `MultiTargetDijkstra`($G_{\text{cand}}^{(u)}$,$u$,$\{v : \sigma_u \leq \sigma_v\}$ in $\sigma(G_j)$)
28: $\quad$ **end for**
29: **end for**
30: **return** `paths`

We implemented a number of speedups to improve the runtime of Algorithm 2, two of which are notable. First, when updating distances in the `dist` dictionary in Line 27, many of the $u \rightsquigarrow v$ pairs can remain untouched if they do not include any nodes from the newly-added `paths`$[j]$. Thus, in the `MultiTargetDijkstra`() function call, we can set the target nodes $T$ to be only the nodes $v$ for which $u \rightsquigarrow v$ needs to be recalculated. Note that we still need to pass in the full set of target nodes $T$, as a fourth argument, to ensure that no nodes in $G_j$ are used as internal nodes on a path. In practice, this dramatically reduced the number of target nodes that `MultiTargetDijkstra`() needed to find, re-running between 0.2%–14% of the possible number of target nodes across all runs.

| | Ground Truth Pathway | | | | DAG Conversion | | |
|---|---|---|---|---|---|---|---|
| **Name** | Nodes | Edges | Receptors | TRs | PL Paths | DAG Nodes | DAG Edges |
| BCR | 137 | 478 | 1 | 18 | 958 | 58 | 131 |
| IL1 | 43 | 442 | 3 | 5 | 713 | 22 | 55 |
| Wnt | 106 | 186 | 14 | 14 | 984 | 40 | 89 |
| EGFR1 | 231 | 1508 | 6 | 33 | >1000 | 105 | 461 |
| TGF$\beta$ | 209 | 900 | 5 | 77 | 70 | 33 | 70 |
| TCR | 154 | 526 | 7 | 20 | 952 | 57 | 139 |

Table 1: Signaling pathways chosen for this study. TRs: transcriptional regulators. PL: PathLinker. The DAG conversion is described in Section 3.1.

Another dramatic speedup comes from avoiding re-computations when minimizing **min_paths_cost**. Lines 16–18 generate a new DAG $G'$, calls `PathCounter`() on this new graph, and uses these values to compute the **min_paths_cost** for every ordered node pair $u, v$. In practice, we first calculate $f$ for $G_j$ as the first step in the main `for` loop, and use functions to update $f$ as if we have added the path $u \rightsquigarrow v$ (without needing to create a new graph $G'$). These update functions, similar to determining the `upstream` and `downstream` dictionaries in Algorithm 1, adjusts all paths from $s$ to $u$ and all paths from $v$ to $t$ assuming that path $u \rightsquigarrow v$ has been added.

# 3 Results

We ran `GrowDAGs` for both cost functions to reconstruct five diverse signaling pathways from the NetPath database [3]. We first define $G_0$ as the shortest $s$-$t$ path in $G$ and compare the topologies of the resulting pathway reconstructions with the PathLinker, the $k$-shortest paths approach [7]. We then define $G_0$ to be a DAG representing the ground truth pathways as represented in NetPath to illustrate how `GrowDAGs` has the potential to suggest additional proteins and interactions for canonical signaling pathways. We identified receptors and transcriptional regulators for each pathway using previously established resources of these proteins types [17, 18, 19]. The five networks range in size, density, and the number of receptors and transcriptional regulators (Table 1). We used an interactome $G$ with $17,513$ nodes and $577,617$ directed edges weighted by experimental evidence [12]. The edge weights are negative log transformed to become edge costs, which we aim to minimize.

## 3.1 $G_0$ as Shortest Paths

We first calculated the shortest path from any receptor to any transcriptional regulator to define $G_0$. Using the Wnt pathway as an example pathway reconstruction, **min_edge_cost** tended to produce DAGs whose paths reuse the same nodes while **min_paths_cost** tended to produce DAGs with more non-overlapping $s$-$t$ paths (Figure 3A & B). Thus, **min_edge_cost** generated DAGs with fewer nodes, fewer edges, and more triangles than DAGs generated by **min_paths_cost** (Figure 3C). These relationships between **min_edge_cost** DAGs and **min_paths_cost** DAGs hold across all six pathway reconstructions; further, a PathLinker reconstructions were often the same size as **min_edge_cost** DAGs but contain very few triangles (Figure 4).

## 3.2 $G_0$ as NetPath DAGs

To illustrate that `GrowDAGs` can begin with any DAG, we then took the ground truth networks and converted them into a $G_0$ DAG in the following way: we calculated up to $1,000$ shortest paths from any receptor to any transcriptional regulator in the NetPath pathway to generate a list of paths.

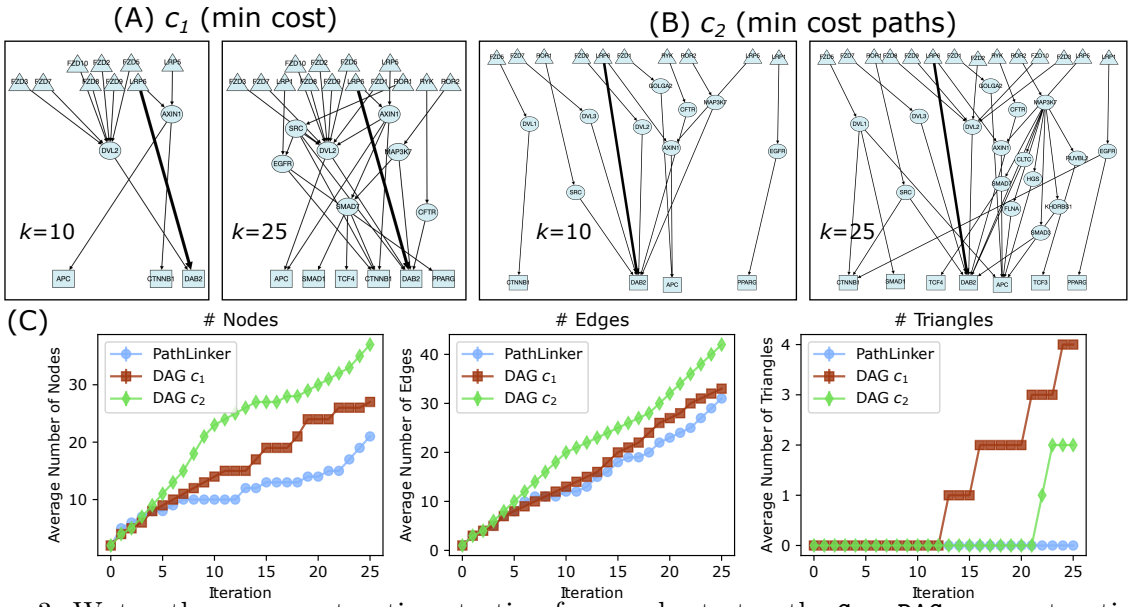


Figure 3: Wnt pathway reconstruction starting from a shortest path. `GrowDAGs` reconstruction for (A) **min_edge_cost** and (B) **min_paths_cost**; $k = 10, 25$ shown, shortest path $G_0$ shown as the thick edge. (C) Number of nodes, edges, and triangles of $G_j$ at each iteration.

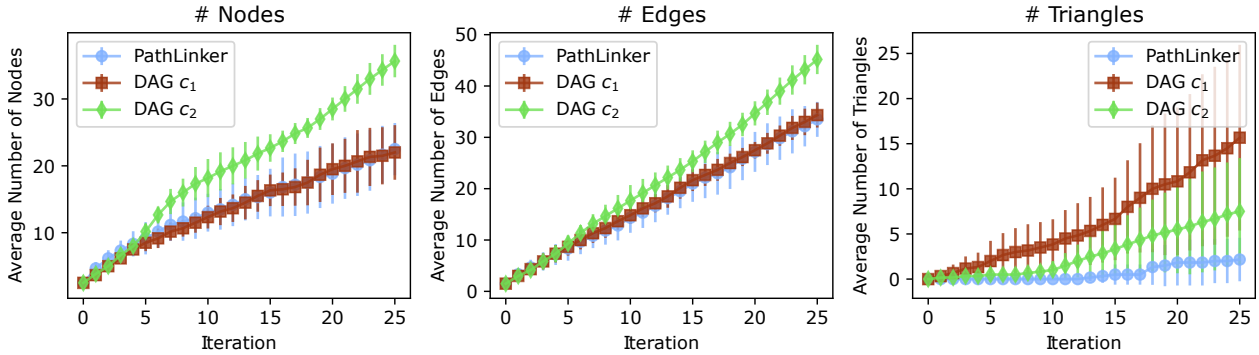


Figure 4: Number of nodes, edges, and triangles of $G_j$ at each iteration averaged across all six signaling pathways.

The union of these paths often included cycles, so we built $G_0$ by adding each path only if it did not introduce a cycle in $G_0$. The total number of nodes and edges in the DAG conversion of $G_0$ also varied in size (Table 1).

Running `GrowDAGs` on the five signaling pathways with the NetPath DAG $G_0$ and $k = 25$ produced similar trends in network topology for **min_edge_cost** and **min_paths_cost** that were shown in Section 3.2. For the 25 new paths added to the NetPath DAGs, we also counted the number of directed edges that were present in Netpath or KEGG pathway databases, indicating that a known signaling interaction was recovered. We found that, on average, $16.9 \pm 3.4$ directed edges from DAGs generated by **min_edge_cost** and $13.2 \pm 3.0$ directed edges from DAGs generated by **min_paths_cost** were present in the databases. An example reconstruction for IL1, the smallest pathway, is shown in Figure 5. These reconstructions potentially suggest new proteins and interactions that should be considered in the annotated pathways, and the percentage of interactions from signaling pathway databases are a promising sign of identifying signaling edges in these DAGs.

# 4 Conclusion

We have described the Growing DAG Problem, a pathway reconstruction formulation that directly adds paths to a DAG that explicitly optimize one of two cost functions (minimizing edge costs or minimizing $s$-$t$ path costs). We presented the `GrowDAGs` algorithm that iteratively finds the most optimal DAG $G_j$. Applying this algorithm to six diverse NetPath pathways, we show that **min_edge_cost** and **min_paths_cost** admits DAGs with different network topologies. Further, we demonstrate that we can begin with a larger DAG $G_0$ such as a DAG converted from the NetPath ground truth. To our knowledge, this is the first pathway reconstruction algorithm that can grow a network based on a seeded subnetwork (rather than a set of nodes).

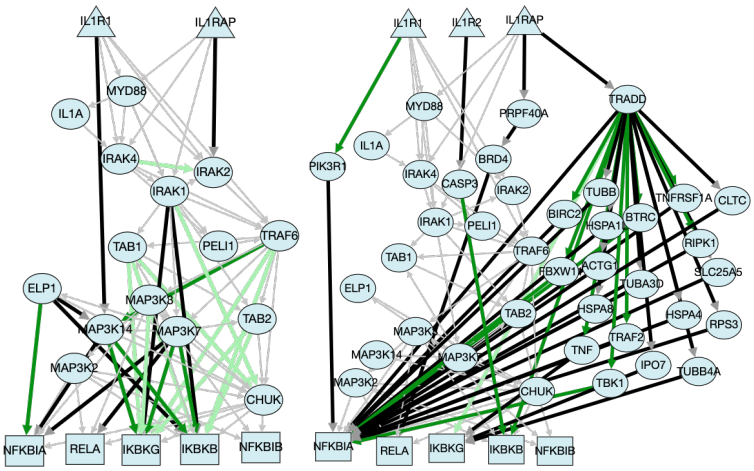


Figure 5: IL1 Reconstructions for (left) **min_edge_cost** and (right) **min_paths_cost** for $k = 25$. Here, $G_0$ is shown in thin gray edges. Dark green edges appear in the NetPath database; light green edges appear in the KEGG database.

While this framework is a promising way to reconstruct signaling pathways with certain topologies, a number of limitations remain that prohibit its widespread use for large pathway reconstructions. First, the algorithm is still slow in practice, namely due to having to check all pairs of topologically sorted nodes of $G_{j-1}$ at each iteration. While we mention a few speedups in Section 2.4, more improvements are needed to reconstruct pathways at the scale of PathLinker (many dozens of nodes and hundreds of edges). Second, this problem formulation and algorithm are limited by the DAG constraints, and reconstructions will never be able to find cycles in pathways. Third, other cost functions may produce different topologies than those reconstructed here; for example, minimizing the average cost of new paths or taking a weighted average of **min_edge_cost** and **min_paths_cost**. However, Lemma 2 does not hold for such cost functions. Finally, while we focus on network topology features in this paper, future work involves assessing the `GrowDAGs` pathway reconstructions in terms of their accuracy in recovering ground truth interactions.

Despite these challenges, the Growing DAG Problem presents a first step towards explicitly optimizing a cost function related to pathway reconstruction. Existing pathway reconstruction methods will always return networks with the same topologies – $k$ shortest paths, Steiner forests, random walks, etc. – and these cost functions provide control over the topologies of the `GrowDAG` reconstructions.


## Acknowledgements

This work was supported by the NSF (DBI-1750981) to AR. We thank Layla Oesper, Ibrahim Youssef, and Tobias Rubel for feedback on early versions of this manuscript.

## References


[1] Antonio Fabregat, Steven Jupe, Lisa Matthews, Konstantinos Sidiropoulos, Marc Gillespie, Phani Garapati, Robin Haw, Bijay Jassal, Florian Korninger, Bruce May, et al. The reactome pathway knowledgebase. *Nucleic acids research*, 46(D1):D649–D655, 2018.

[2] Minoru Kanehisa, Michihiro Araki, Susumu Goto, Masahiro Hattori, Mika Hirakawa, Masumi Itoh, Toshiaki Katayama, Shuichi Kawashima, Shujiro Okuda, Toshiaki Tokimatsu, et al. Kegg for linking genomes to life and the environment. *Nucleic acids research*, 36(suppl_1):D480–D484, 2007.

[3] Kumaran Kandasamy, S Sujatha Mohan, Rajesh Raju, Shivakumar Keerthikumar, Ghantasala S Sameer Kumar, Abhilash K Venugopal, Deepthi Telikicherla, J Daniel Navarro, Suresh Mathivanan, Christian Pecquet, et al. Netpath: a public resource of curated signal transduction pathways. *Genome biology*, 11(1):1–9, 2010.

[4] Alexander R Pico, Thomas Kelder, Martijn P Van Iersel, Kristina Hanspers, Bruce R Conklin, and Chris Evelo. Wikipathways: pathway editing for the people. *PLoS biology*, 6(7):e184, 2008.

[5] Ethan G Cerami, Benjamin E Gross, Emek Demir, Igor Rodchenkov, Özgün Babur, Nadia Anwar, Nikolaus Schultz, Gary D Bader, and Chris Sander. Pathway commons, a web resource for biological pathway data. *Nucleic acids research*, 39(suppl_1):D685–D690, 2010.

[6] Damian Szklarczyk, Annika L Gable, Katerina C Nastou, David Lyon, Rebecca Kirsch, Sampo Pyysalo, Nadezhda T Doncheva, Marc Legeay, Tao Fang, Peer Bork, et al. The string database in 2021: customizable protein–protein networks, and functional characterization of user-uploaded gene/measurement sets. *Nucleic acids research*, 49(D1):D605–D612, 2021.

[7] Anna Ritz, Christopher L Poirel, Allison N Tegge, Nicholas Sharp, Kelsey Simmons, Allison Powell, Shiv D Kale, and TM Murali. Pathways on demand: automated reconstruction of human signaling networks. *NPJ systems biology and applications*, 2(1):1–9, 2016.

[8] Nurcan Tuncbag, Alfredo Braunstein, Andrea Pagnani, Shao-Shan Carol Huang, Jennifer Chayes, Christian Borgs, Riccardo Zecchina, and Ernest Fraenkel. Simultaneous reconstruction of multiple signaling pathways via the prize-collecting steiner forest problem. *Journal of computational biology*, 20(2):124–136, 2013.

[9] Jochen Supper, Lucía Spangenberg, Hannes Planatscher, Andreas Dräger, Adrian Schröder, and Andreas Zell. Bowtiebuilder: modeling signal transduction pathways. *BMC systems biology*, 3(1):1–13, 2009.

[10] Taher H Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE transactions on knowledge and data engineering*, 15(4):784–796, 2003.

[11] Alex Lan, Ilan Y Smoly, Guy Rapaport, Susan Lindquist, Ernest Fraenkel, and Esti Yeger-Lotem. Responsenet: revealing signaling and regulatory networks linking genetic and transcriptomic screening data. *Nucleic acids research*, 39(suppl_2):W424–W429, 2011.

[12] Ibrahim Youssef, Jeffrey Law, and Anna Ritz. Integrating protein localization with automated signaling pathway reconstruction. *BMC bioinformatics*, 20(16):1–15, 2019.

[13] Tobias Rubel and Anna Ritz. Augmenting signaling pathway reconstructions. In *Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 1–10, 2020.

[14] Mitchell J Wagner, Aditya Pratapa, and TM Murali. Reconstructing signaling pathways using regular language constrained paths. *Bioinformatics*, 35(14):i624–i633, 2019.

[15] Jin Y Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

[16] Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[17] Markus Sällman Almén, Karl JV Nordström, Robert Fredriksson, and Helgi B Schiöth. Mapping the human membrane proteome: a majority of the human membrane proteins can be classified according to function and evolutionary origin. *BMC biology*, 7(1):1–14, 2009.

[18] Timothy Ravasi, Harukazu Suzuki, Carlo Vittorio Cannistraci, Shintaro Katayama, Vladimir B Bajic, Kai Tan, Altuna Akalin, Sebastian Schmeier, Mutsumi Kanamori-Katayama, Nicolas Bertin, et al. An atlas of combinatorial transcriptional regulation in mouse and man. *Cell*, 140(5):744–752, 2010.

[19] Juan M Vaquerizas, Sarah K Kummerfeld, Sarah A Teichmann, and Nicholas M Luscombe. A census of human transcription factors: function, expression and evolution. *Nature Reviews Genetics*, 10(4):252–263, 2009.