Growing DAGs: Optimization Functions for Pathway Reconstruction Algorithms

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

[1]Computer Science Department, Reed College, Portland, OR, USA
[2]Biology Department, Reed College, Portland, OR, USA,

[†]These Authors Contributed Equally.
[◁]Current Affiliation: School of Science Science, Aalto University
[▷]Current Affiliation: Microsoft Corporation
[*]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 evaluate the pathway reconstructions when applied to six diverse signaling pathways from the NetPath database. Growing DAGs is a promising step towards reconstructing pathways that provably optimize a specific cost function.

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 [6], 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 [7] and network flow [8]). 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) [9] 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 canonical Wnt signaling [6].

Despite PathLinker's success, calculating the first 20,000 shortest paths from receptors to transcriptional regulators for each signaling pathway in the NetPath databases captured only about 70% of the known signaling interactions [6], leaving much room for improvement. Extensions of PathLinker have used auxiliary data such as protein localization information [10] or additional downstream processing [11] to accurately reconstruct ground truth pathways. Other extensions have constrained the paths from R to T to follow regular expression patterns [12].

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 two pathway reconstruction methods that solve this new variant of the problem and apply them to six signaling pathways from the NetPath database [3]. We show that DAG reconstructions exhibit different topologies, they often outperform a previous reconstruction method in recovering annotated proteins and interactions, and the predicted nodes are enriched for different biological processes. 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

We now formulate an optimization problems to grow directed acyclic graphs (DAGs) from a graph G. 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, directed 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.

The shortest-path aspect of this problem comes from minimizing the cost functions. Note that, while KSP approaches will always calculate the jth shortest s-t path in G, this path may be comprised of edges already in G_{j-1} and thus fail the first condition above. KSP approaches are also not guaranteed to produce a DAG at each iteration.

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. Given G_{j-1} and G_j, we can efficiently calculate the cost functions described above. 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

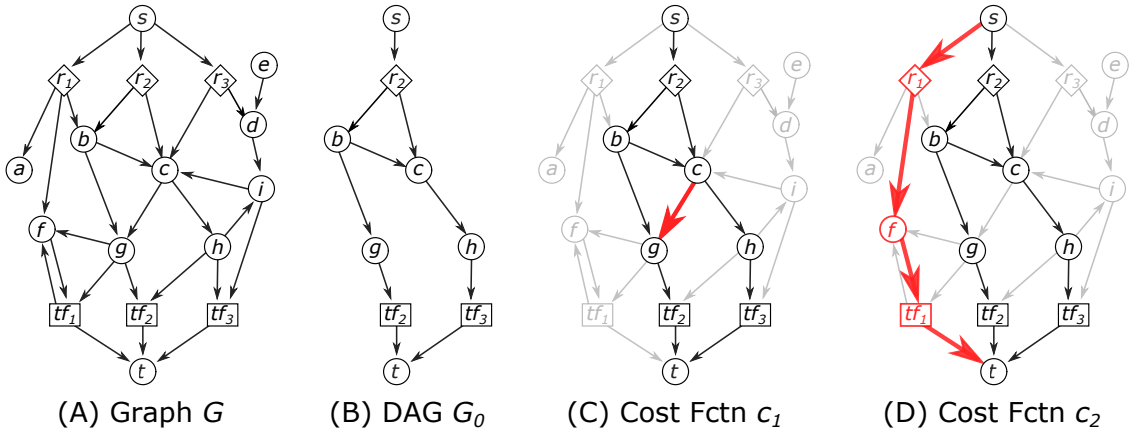

Figure 1: The Growing DAG Problem. (A) Input graph G which may contain cycles and bidirected edges. Node s is connected to three receptors r_1, r_2, r_3; three transcription factors tf_1, tf_2, tf_3 are connected to node t. In this example, all edges have unit cost. (B) An example DAG $G_0 \subset G$. (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.

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[1] [13], 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 $(u,v) \in E_j$. Supplementary Section S1.1 describes the `PathCounter`() algorithm, which returns f_{uv} for every edge (u,v) in a graph.. Once we have f from the `PathCounter`() algorithm, 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_Δ will also be a DAG (which may be disconnected). We first prove some properties of G_Δ that hold for either of the cost functions listed.

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_Δ that minimizes any of the cost functions will be exactly one path.*

Proof. G_Δ is a non-empty DAG, since at least one new s-t path must exist in G_j. If G_Δ is not exactly one path, then G_Δ 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_Δ 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_Δ.

[1] #P (Sharp-P) problems are the counting problems associated with decision problems in *NP*.

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_Δ 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_Δ. 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_Δ 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_Δ is not optimal.

□

Lemma 2. *G_Δ 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_Δ 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_Δ 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_Δ.

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_Δ 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_Δ is not optimal.
2. $x = v_0$. Suppose x is some node on the path p that is not v_0; call it v_i. 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 σ_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 1 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_{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 the `PathCounter`() algorithm (Supplementary Algorithm S1) 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.

We implemented a number of speedups to improve the runtime of Algorithm 1, 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.

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 the `PathCounter`() algorithm, adjusts all paths from s to u and all paths from v to t assuming that path $u \rightsquigarrow v$ has been added.

Algorithm 1 `GrowDAGs`($G = (V, E)$,$G_0 = (V_0, E_0)$, num iters k, cost function c)

```
paths ← []
dist ← {}
# Initialize distances
G_cand ← G \ G_0
for u ∈ V_0 do
    G_cand^(u) ← Remove any node v from G_cand where σ_u > σ_v in σ(G_0)
    dist[u] = MultiTargetDijkstra(G_cand^(u),u,{v : σ_u ≤ σ_v} in σ(G_0))
end for
for j = 1, 2, ..., k do
    # Get the min cost path according to cost function c
    if c = c_1 then
        paths[j] ← the path u ⇝ v from dist[u][v] such that min_{x,y} dist[x][y] =dist[u][v]
    else if c = c_2 then
        C ← {}
        for u, v such that σ_u ≤ σ_v in σ(G_j) do
            G' ← Add the path u ⇝ v from dist[u][v] to G_{j-1}
            f ← PathCounter(G',s,t)
            C[u][v] = Σ_{(u,v)∈E'} f_uv w_uv
        end for
        paths[j] ← the path u ⇝ v from dist[u][v] such that min_{x,y} C[x][y] = C[u][v]
    end if
    # Make G_j and update distances
    G_j ← Add paths[j] to G_{j-1}
    G_cand ← G \ G_j
    for u ∈ V_j do
        G_cand^(u) ← Remove any node v from G_cand where σ_u > σ_v in σ(G_j)
        dist[u] = MultiTargetDijkstra(G_cand^(u),u,{v : σ_u ≤ σ_v} in σ(G_j))
    end for
end for
return paths
```

3 Results

We ran `GrowDAGs` for both cost functions to reconstruct six diverse signaling pathways from the NetPath database [3]. We identified receptors and transcriptional regulators for each pathway using previously established resources of these proteins types [14, 15, 16]. The six 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 [10]. The edge weights are negative log transformed to become edge costs, which we aim to minimize.

We first defined 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 [6]. We ran `GrowDAGs` for each cost function on each of the six pathways for $k = 200$ starting from the shortest-path G_0.

3.1 Topological Differences between DAG Reconstructions

We first provide some examples of reconstructed pathways using `GrowDAGs` by examining the topological differences between DAG reconstructions of the Wnt signaling pathway. As expected, **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 2A & B shows $k = 75$ for the

	Ground Truth Pathway				DAG Conversion		
Name	Nodes	Edges	Receptors	TRs	PL Paths	DAG Nodes	DAG Edges
BCR	137	456	1	18	958	58	131
EGFR1	231	1456	6	33	>1000	105	461
IL1	43	178	3	5	713	22	55
TCR	154	504	7	20	952	57	139
TGFβ	209	863	5	77	70	33	70
Wnt	106	428	14	14	984	40	89

Table 1: Signaling pathways chosen for this study. TRs: transcriptional regulators. PL: PathLinker. The DAG conversion is described in Section 3.4, where they are used as G_0.

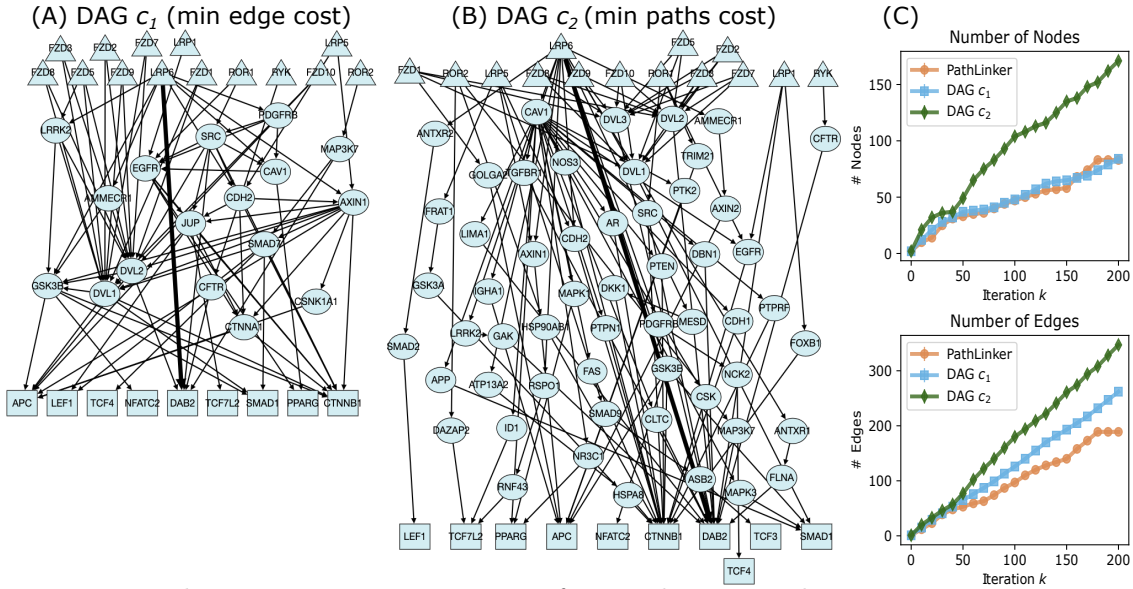

Figure 2: Wnt pathway reconstruction starting from a shortest path. `GrowDAGs` reconstruction for (A) **min_edge_cost** and (B) **min_paths_cost**; $k = 75$ shown, shortest path G_0 shown as the thick edge. (C) Number of nodes, edges, and feed forward loops (FFLs) of G_j at each iteration. c_1: **min_edge_cost**; c_2: **min_paths_cost**.

Wnt pathway reconstruction). Cost function **min_edge_cost** generated DAGs with fewer nodes and fewer edges than DAGs generated by **min_paths_cost** (Figure 2C). 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 (Supplementary Figure S2). Network visualizations of all pathways up to $k = 200$ are available on GraphSpace [17]: http://graphspace.org/graphs/?query=tags:DAG#.

3.2 Comparison to Ground Truth Pathways

We then evaluated how well the `GrowDAG` pathways captured annotated proteins and interactions from the six NetPath pathways. For this analysis, we considered the NetPath pathways ground truth nodes and edges, though it is widely acknowledged that signaling pathway databases are incomplete. For each of the six pathways, we calculated the precision and recall of nodes and edges as we increase k for DAG **min_edge_cost**, DAG **min_paths_cost**, and PathLinker reconstructions for k up to 200 (Figure 3 and Supplementary Figure S3). When computing precision and recall

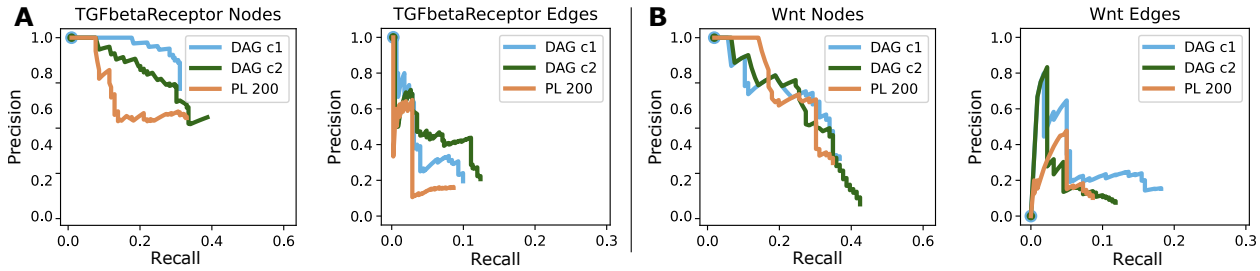

Figure 3: Precision-recall curves for reconstructions from two pathways: (A) TGFβ and (B) Wnt signaling. Node and edge precision-recall are shown on the left and right of each sub-panel. c1: **min_edge_cost**; c2: **min_paths_cost**. Precision-recall curves for all six pathways is shown in Supplementary Figure S3.

of edges, we compare undirected edges to undirected ground truth edges, which is consistent with previous evaluations [6]. Overall, the first 200 iterations of the methods achieve relatively low recall (e.g., $0.2 - -0.6$ recall for nodes and $0.05 - -0.3$ recall for edges) despite relatively high precision for edges (Supplementary Figure S3). DAG **min_paths_cost** achieves the higher AUPRC for all node evaluations and for four of the six edge evaluations; further, PathLinker outperforms the DAG reconstructions on TCR, which has an extremely low AUPRC across the board (Table 2).

While the methods have low recall, $k = 200$ begins to reach the limit of useful network visualization (for example, the networks visualized in Figure 2A & B are at $k = 75$). Instead of evaluating the AUPRC, considering the precision for the top 50 predictions (e.g., the first 50 nodes or first 50 edges in a reconstruction) is a more realistic assessment in terms of reconstruction usefulness. When considering the first 50 predictions, DAG **min_paths_cost** is no longer consistently outperforms the other two methods, though one of the `GrowDAG` reconstructions has the highest precision for five of the node reconstructions and five of the edge reconstructions (Table 3).

One of the reasons for the low precision, especially for edges, is that the number of negative examples is vastly larger than the number of positive examples for a single pathway. Thus, previous evaluation frameworks have subsampled the negative examples from the interactome to ensure a 50:1 ratio of negatives to positives [6]). This inflates the precision but can also more clearly separate performances of different approaches. We conducted the same precision-recall analysis by subsampling negatives in a 50:1 ratio and repeating for 10 iterations – the trends above hold even when subsampling negatives, including the precision of the top 25 predictions (see Supplementary Figure S4 and Supplementary Tables S1–S2).

Name	**Node AUPRC**			**Edge AUPRC**		
	DAG c_1	**DAG c_2**	**PL**	**DAG c_1**	**DAG c_2**	**PL**
BCR	0.104	**0.255**	0.129	0.015	**0.028**	0.002
EGFR1	0.180	**0.205**	0.146	0.007	**0.010**	0.005
IL1	0.148	**0.316**	0.268	0.020	**0.052**	0.035
TCR	0.117	**0.187**	0.161	0.003	0.003	**0.005**
TGFβ	0.295	**0.318**	0.241	0.043	**0.057**	0.024
Wnt	0.274	**0.292**	0.268	**0.056**	0.029	0.021

Table 2: Area under the precision-recall (AUPRC) values for nodes (left) and edges (right) for the reconstructions at $k = 200$. c_1: **min_edge_cost**; c_2: **min_paths_cost**. Largest AUPRC values are shown in bold.

Name	Node Precision @50			Edge Precision @50		
	DAG c_1	DAG c_2	PL	DAG c_1	DAG c_2	PL
BCR	0.580	**0.720**	0.392	0.040	**0.231**	0.040
EGFR1	**0.880**	0.725	0.660	0.120	**0.180**	0.100
IL1		**0.380**	0.320	0.120	0.137	**0.157**
TCR	0.480	0.615	**0.640**	**0.120**	0.020	0.118
TGFβ	**0.960**	0.860	0.640	0.360	**0.440**	0.255
Wnt	**0.660**	0.627	0.640	**0.240**	0.200	0.220

Table 3: Precision of the first 50 predictions (nodes on the left, edges on the right) for the reconstructions. c_1: **min_edge_cost**; c_2: **min_paths_cost**. Largest values are shown in bold; missing entries denote that 50 unique predictions weren't reached at $k = 200$.

3.3 Reconstructions Capture Diverse Biological Processes

Next, we evaluated whether the three pathway reconstruction methods (DAG **min_edge_cost**, DAG **min_paths_cost**, and PathLinker) recover different biological processes. Here, we considered the nodes in the reconstructions and found that each reconstruction often contains distinct nodes not found in the other reconstructions. For example, only 15% of the predicted nodes for Wnt $k = 200$ reconstructions appear in all reconstructions (Figure 4; all pathways shown in Supplementary Figure S5). We performed gene function enrichment using PantherDB [18, 19] on the uniquely-predicted nodes for each method (e.g., 38 DAG **min_edge_cost** nodes, 113 DAG **min_paths_cost** nodes, and 28 PathLinker nodes in Figure 4) to determine whether any Panther pathways are enriched in each of these unique sets.

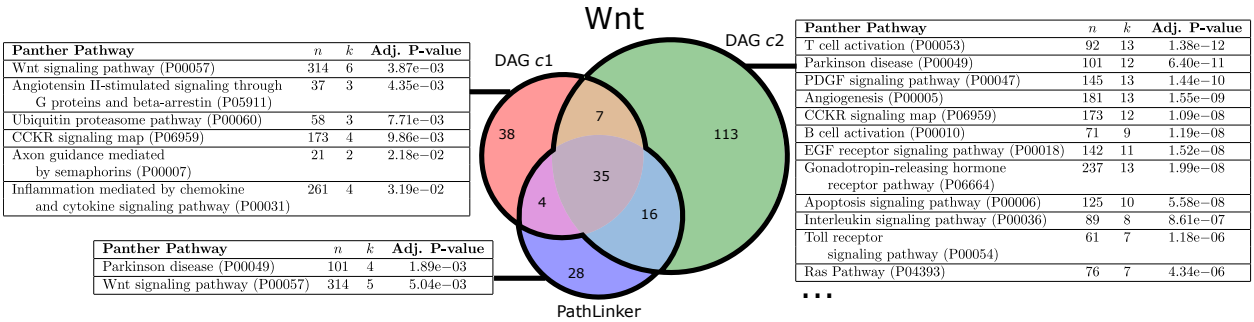

Figure 4: Venn diagram of predicted nodes for DAG **min_edge_cost**, DAG **min_paths_cost**, and PathLinker for Wnt reconstructions $k = 200$. For set of nodes uniquely predicted by each algorithm, we ran the PANTHER Over-representation Test (PANTHER version 17.0, Fisher test with FDR correction of 0.05). n: number of proteins in the Panther pathway; k: number of predicted proteins in the Panther pathway; c1: **min_edge_cost**; c2: **min_paths_cost**. Full table for DAG **min_paths_cost** is shown in Supplementary Table S3.

We found that each set of unique nodes were enriched for at least two Panther pathways (Tables in Figure 4). DAG **min_paths_cost** has 31 enriched Panther pathways; the full table is in Supplementary Table S3. All three reconstructions contain nodes that are enriched in Wnt signaling (DAG **min_paths_cost** captures ten nodes with an adjusted p-value of 8.74×10^{-5}, Supplementary Table S3), each capturing different aspects of the Wnt pathway that is separate from the other two methods or even the nodes the methods collectively predict. DAG **min_paths_cost** and PathLinker reconstructions are also enriched for Parkinson Disease, which has a known connection

to Wnt signaling through inflammatory pathways [20]. DAG **min_paths_cost** also appears to be enriched in many signaling pathways that are known to cross-talk with Wnt, including T cell signaling [21], B cell signaling [22], and EGFR signaling [23], among others.

3.4 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. 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 3.1. 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 ± 3.4 directed edges from DAGs generated by **min_edge_cost** and 13.2 ± 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.

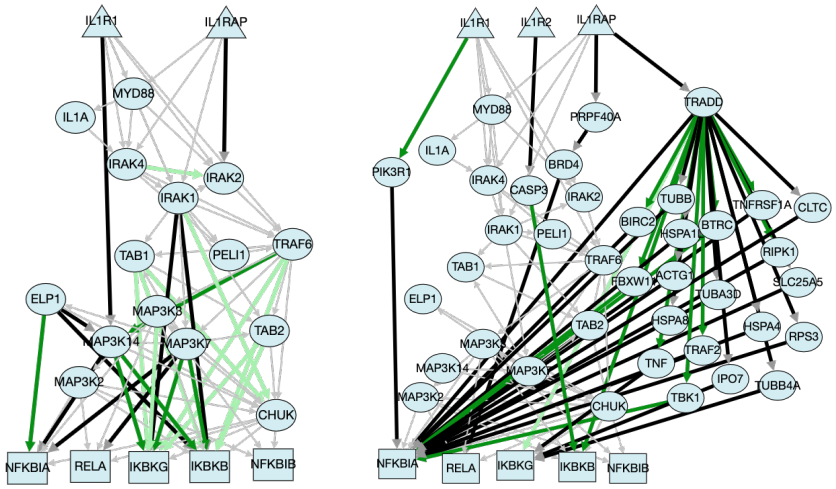

Figure 5: IL1 Reconstructions for (left) **min_edge_cost** and (right) **min_paths_cost** for $k = 25$. Here, G_0 is shown in thin gray edges. Dark green edges appear in the NetPath database; light green edges appear in the KEGG database.

4 Discussion

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 that often outperforms a KSP approach compared to the annotated proteins and interactions in NetPath. Further, the reconstructions return proteins that are enriched for different biological processes. We also 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).

The Growing DAG Problem and subsequent algorithms are limited by the DAG constraints – namely, that reconstructions will never contain cycles. Feedback loops are an integral part of signaling [24], and many of the signaling pathways in NetPath, KEGG, and other pathway databases have documented examples of these cycles. Our reasons for starting with DAGs were twofold: first, intracellular signaling that begins at a cell membrane and ends with transcriptional regulation has a general direction; and second, DAGs provide a reduced search space when growing networks. Beyond the search space for optimal subgraphs at each iteration, the **min_paths_cost** optimization criterion could not be efficiently computed on a subgraph with cycles, though there may very well be another optimization function that would be useful to consider in general graphs. Generalizing the computational problem and the general framework to graphs with cycles would be an important future step in pathway reconstruction.

Both of the `GrowDAG` formulations as well as PathLinker rely on a parameter k, which determines how large to grow the pathway reconstruction. For this work, we chose to use $k = 200$, which begins to reach the limit of useful network visualization (interactive networks are available at http://graphspace.org/graphs/?query=tags:DAG#). However, this small value of k limits reconstructions to very low recall compared to ground truth pathways (Figure 3 and Table 2). By increasing k, the precision-recall curves will achieve larger recall values. In previous work, PathLinker required $k = 20,000$ paths to achieve an average recall of 0.7 [6]. We anticipate that the DAG formulations would require fewer iterations to achieve a similar recall value, since each iteration of our method is guaranteed to add new paths. A full comparison of precision and recall for large values of k is beyond the scope of this work, but would be valuable to conduct.

While the `GrowDAGs` framework is a promising way to reconstruct signaling pathways with certain topologies, a few 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. Runtimes for $k = 100$ averaged 2.1 hours, ranging from eleven minutes to 16 hours (a full table of runtimes is available in Supplementary Table S4). However, computing the next iterations for $k = 101, \ldots, 200$ averaged 19.6 hours, ranging from 44 minutes to nearly 52 hours. 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, 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.

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 the proposed cost functions provide control over the topologies of the `GrowDAG` reconstructions. This work opens the door for improved cost functions that reflect ground truth pathways and provides a framework for designing algorithms to grow networks.

Data and Code Availability

All code and data related to Growing DAGs are available on GitHub: https://github.com/Reed-CompBio/growing-dags. Pathway reconstruction visualizations for the six pathways and $k = 200$ are available on GraphSpace [17]: http://graphspace.org/graphs/?query=tags:DAG#.

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] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] Aditya Bharadwaj, Divit P Singh, Anna Ritz, Allison N Tegge, Christopher L Poirel, Pavel Kraikivski, Neil Adames, Kurt Luther, Shiv D Kale, Jean Peccoud, et al. Graphspace: stimulating interdisciplinary collaborations in network biology. *Bioinformatics*, 33(19):3134–3136, 2017.

[18] Huaiyu Mi and Paul Thomas. Panther pathway: an ontology-based pathway database coupled with data analysis tools. In *Protein networks and pathway analysis*, pages 123–140. Springer, 2009.

[19] Huaiyu Mi, Anushya Muruganujan, Xiaosong Huang, Dustin Ebert, Caitlin Mills, Xinyu Guo, and Paul D Thomas. Protocol update for large-scale genome and gene function analysis with the panther classification system (v. 14.0). *Nature protocols*, 14(3):703–721, 2019.

[20] Francesca L'Episcopo, Cataldo Tirolo, Salvo Caniglia, Nuccio Testa, Maria Concetta Morale, Maria Francesca Serapide, Stefano Pluchino, and Bianca Marchetti. Targeting wnt signaling at the neuroimmune interface for dopaminergic neuroprotection/repair in parkinson's disease. *Journal of molecular cell biology*, 6(1):13–26, 2014.

[21] Xin Li, Yanwei Xiang, Fulun Li, Chengqian Yin, Bin Li, and Xisong Ke. Wnt/β-catenin signaling pathway regulating t cell-inflammation in the tumor microenvironment. *Frontiers in immunology*, 10:2293, 2019.

[22] Bin Ma and Michael O Hottiger. Crosstalk between wnt/β-catenin and nf-κb signaling pathway during inflammation. *Frontiers in immunology*, 7:378, 2016.

[23] Tianhui Hu and Cunxi Li. Convergence between wnt-β-catenin and egfr signaling in cancer. *Molecular cancer*, 9(1):1–7, 2010.

[24] Uri Alon. Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8(6):450–461, 2007.