# A Framework for Path-Oriented Network Simplification

Hannu Toivonen, Sébastien Mahler, and Fang Zhou

Department of Computer Science and
Helsinki Institute for Information Technology HIIT,
PO Box 68, FI-00014 University of Helsinki, Finland
`firstname.lastname@cs.helsinki.fi`

**Abstract.** We propose a generic framework and methods for simplification of large networks. The methods can be used to improve the understandability of a given network, to complement user-centric analysis methods, or as a pre-processing step for computationally more complex methods. The approach is path-oriented: edges are pruned while keeping the original quality of best paths between all pairs of nodes (but not necessarily all best paths). The framework is applicable to different kinds of graphs (for instance flow networks and random graphs) and connections can be measured in different ways (for instance by the shortest path, maximum flow, or maximum probability). It has relative neighborhood graphs, spanning trees, and certain Pathfinder graphs as its special cases. We give four algorithmic variants and report on experiments with 60 real biological networks. The simplification methods are part of ongoing projects for intelligent analysis of networked information.

## 1   Introduction

In many fields, information is naturally represented as networks of interlinked people, genes, computers, web pages, concepts, or other objects. Networks or graphs are a simple yet powerful formalism, but for finding or extracting the most relevant parts of large networks, intelligent data analysis can be useful.

As an example, consider biological networks and their analysis. Public biological databases are interlinked and form a huge graph of biological concepts. Intelligent query mechanisms allow users to extract smaller subgraphs, maximally relevant to user-specified query nodes [1, 2]. In research efforts such as Bison [3], heterogeneous information repositories are merged to large networks which are then analyzed and explored with intelligent methods.

In this paper, we propose a generic framework and methods for simplification of large, weighted networks. The proposed methods can be used to improve the understandability of a given graph, to complement user-centric analysis methods, or as a pre-processing step for computationally more complex methods.

A property of primary interest in many applications is the strength of an (indirect) connection between some given nodes. Such measures are crucial, for example, in link prediction. The proposed framework builds on the assumption

that connections between nodes are measured using the best path between them and then removes edges while *keeping the original quality of connection for all pairs of nodes.*

The elegance of the proposed solution is in its wide genericity. It is applicable to different kinds of graphs (for example, flow networks and random graphs) and connectivity can be measured in different ways (for example, by the shortest path, maximum flow, or maximum probability). The proposed framework has both relative neighborhood graphs and spanning trees as its special cases.

On the other hand, the assumption that the connection between nodes is measured with the best path between them is quite strong. This assumption allows, however, a clear specification of the result without reference to an algorithm that computes it. Since the proposed methods are computationally efficient, they may be a practical compromise also in applications where more complex measures of connectivity would otherwise be preferred. We will address this in the experimental section.

The remainder of this article is organized as follows. We first formalize the problem of path-oriented network simplification in Section 2. In Section 3 we present a range of algorithms to prune redundant edges. We briefly review related work in Section 4. We present and discuss experimental results in Section 5 and finally draw some conclusions in Section 6.

## 2   Path-oriented edge and graph redundancy

Our goal is to simplify a given weighted graph $G = (V, E)$ by removing redundant edges. In this section we define the necessary notations and concepts, we give a simple theorem that allows efficient pruning, and we give some examples.

### 2.1   Definitions

We assume a weighted graph $G = (V, E)$ is given. In the following, we assume $G$ is undirected, but the definitions and methods can be easily generalized for directed graphs. An *edge* $e \in E$ is a pair $e = \{u, v\}$ of nodes in $V$. Each edge has a *weight* $w(e) \in \mathbb{R}$. A *path* $P$ is a set of edges $P = \{\{u_1, u_2\}, \{u_2, u_3\}, \dots, \{u_{k-1}, u_k\}\} \subset E$. We use the notation $u_1 \overset{P}{\rightsquigarrow} u_k$ to say that $P$ is a path between $u_1$ and $u_k$, or, equivalently, to say that $u_1$ and $u_k$ are the endvertices of $P$.

In order to allow different definitions of redundancy, we parameterize our problem and methods with a *path quality function* $q : \{P \mid P \text{ is a path in } G\} \to \mathbb{R}$. Without loss of generality, we assume that larger values of $q$ indicate higher quality; this assumption can be easily reversed if necessary.

A natural and simple way to measure relatedness between two nodes $u$ and $v$ in a weighted graph is to consider the quality of the best path between them. This *best path quality* $Q$ is thus defined as $Q(u, v; E) = \max_{P \subset E : u \overset{P}{\rightsquigarrow} v} q(P)$. If $u$ and $v$ are not connected, $Q(u, v; E) = -\infty$.

In this paper, we will study the problem of pruning a graph without affecting the best path quality for *any* pair of nodes. We say that an edge $e \in E$ is *redundant* if and only if

$$Q(u,v;E) = Q(u,v;E \setminus \{e\}) \quad \text{for all } \{u,v\} \in V, \tag{1}$$

in other words, if the best path quality $Q$ does not depend on edge $e$ for *any* nodes $u, v$ in $G$.

Figure 1 illustrates the idea in a small graph. Here edges have lengths, and the length of an edge or a path corresponds to the length of the edge or path in the drawing of Figure 1 (i.e., path length is not defined by the number of edges in it). Now, while there is a direct edge $e = \{u, v\}$ between nodes $u$ and $v$, there exists a shorter path between them. Since edge $e$ does actually not belong to the shortest path between any pair of nodes in the graph, pruning it has no effect on any shortest path.
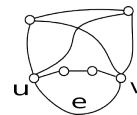


Fig. 1: Edge $e$ is redundant.

A *non-redundant graph* is a graph that contains no redundant edges. We define a *completely pruned graph* as a non-redundant graph that maintains the best path quality of the original graph for all its node pairs. Formally: given a graph $G = (V,E)$, a completely pruned graph $H = (V,F)$ is one where $F \subseteq E$, and $Q(u,v;E) = Q(u,v;F)$ for all $u, v \in V$, and $H$ is non-redundant.

A couple of notes about pruning redundant edges that are in order, especially for the case in which node weights and path qualities may have ties: First, the trivial approach of taking the union of the best paths between all pairs of nodes does then not necessarily lead to a completely pruned graph, since ties might be solved differently for different pairs of nodes; second, a given graph may not have a unique completely pruned subgraph.

## 2.2    Monotone path quality functions

Many path quality measures $q$ are monotone in the sense that replacing a segment of a path by a better one with respect to $q$ never decrease the quality of the whole path. Formally, let $R$ and $S$ be two paths with identical endvertices. Function $q$ is *monotone* if for any path $P$ containing $R$ as a subpath, i.e., $P \supset R$, we have

$$q(R) \leq q(S) \Rightarrow q(P) \leq q(P \setminus R \cup S). \tag{2}$$

Monotonicity is a natural property for many path quality functions but not for all. Consider, for instance, average edge weight. A path consisting of edges with weights $10, 2, 1, 2, 10$ has a smaller average weight ($5$) than an alternative path with weights $10, 1, 10$ (average weight $7$), even though the average of $2, 1, 2$ is larger than $1$.

Monotonicity of path quality allows efficient pruning, as shown by the following theorem. We omit the proof for brevity.

**Theorem 1.** *Let $G = (V,E)$ be a graph and $e \in E$ an edge with endvertices $w_1$ and $w_2$, in other words $w_1 \overset{\{e\}}{\rightsquigarrow} w_2$. Edge $e$ is redundant if and only if there*

*exists another path $S \subset E : w_1 \overset{S}{\leadsto} w_2, e \notin S$, between the endvertices such that $q(S) \geq q(e)$.*

Theorem 1 says that redundancy of an edge (such as $e$ in Figure 1) with respect to *all* pairs of nodes is equivalent to its redundancy with respect to its own endvertices. This means that edge redundancy can be evaluated efficiently by just checking if there is a path between the endvertices ($u$ and $v$) which is at least as good as the direct edge. Redundancy with respect to all other pairs of nodes follows.

The following corollary is a special case of Theorem 1 for paths of length two.

**Corollary 1.** *Let $G = (V, E)$ be a graph and $e = \{u, v\}$ an edge in $E$. Edge $e$ is redundant if there exists a path $S = \{\{u, w\}, \{w, v\}\}$ such that $q(S) \geq q(e)$.*

Corollary 1 gives an even faster way of identifying some redundant edges: for each edge $e$, find the shared neighbors $w$ of the endvertices $u, v$ and check if the path from $u$ via $w$ to $v$ is at least as good as edge $e$ itself. Obviously, this test cannot be used to infer *non*-redundancy of edges.

In Section 3 we will present algorithms that use Theorem 1 and Corollary 1 to solve the pruning problem with different trade-offs between completeness of pruning and time complexity.

Next, however, we give examples of different path quality functions $q$ one could use in specific graph contexts.

### 2.3    Example instances of the framework

*Flow networks* are graphs where edge weights define their capacities. The maximum capacity $c(P)$ of a path is limited by each of its edges: $c(P) = \min_{\{u,v\} \in P} w(\{u, v\})$. The use of $c$ as path quality function would guarantee that the amount of flow that the best path has is always conserved. The global property of network flow is much more commonly used; we will return to this in the experimental section.
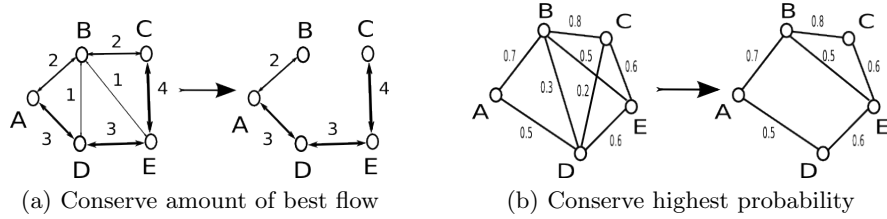


(a) Conserve amount of best flow       (b) Conserve highest probability

Fig. 2: Simplification of different types of graphs

Using the path capacity $c$ above as path quality function actually has the property that completely pruned graphs are *spanning trees* that maximize the smallest edge weight in the tree (Figure 2a). Arbitrary spanning trees could be obtained by simply defining $q(P) = 1$, in other words, $q$ indicates that the endpoints of path $P$ are connected.

As a more complex example, assume that the weight $w(e) \in ]0, 1]$ of an edge is the probability that the connection exists, and that edges are mutually independent. Such graphs are called *probabilistic* or *Bernoulli random graphs*. The best path is the one that most likely exists; the probability of a path $P$ is simply the probability that all of its edges co-exist: $q(P) = \prod_{\{u,v\} \in P} w(\{u, v\})$. Figure 2b illustrates network simplification with this path quality function.

*Shortest paths* are commonly looked for in networks where edge weights correspond to distances between nodes. This is an example of a setting where the path quality—in this case path length—should be minimized rather than maximized. It is easy to change the definitions of this paper. Alternatively, one can transform the weights $w(e)$ to $w'(e) = e^{-w(e)}$ and use the same multiplicative $q$ that was defined above for probabilistic networks.

A *relative neighborhood graph* discards those edges whose endvertices have a shared neighbor that is closer to both endvertices than the endvertices are to each other. Again, edge and path qualities are measured by distances, so we need to minimize $q(P) = \max_{\{u,v\} \in P} w(\{u, v\})$. An alternative is to transform $w'(e) = 1/w(e)$ and use the path quality function $q$ defined above for flow networks. In either case, paths of at most two edges should be considered (cf. Corollary 1).

## 3   Algorithms

### 3.1   Variations

The four algorithmic variants we next present result from two axes of variation: (A) whether to use Theorem 1 or Corollary 1 to search and identify redundancies, and (B) whether to perform iterative search or static search.

*Strategies to identify redundancy* Based on Theorem 1, the *global best path search* determines the redundancy of an edge by finding and evaluating the best path connecting its endpoints (and not including the edge itself). By the theorem, global best path search guarantees correct identification of redundancy.

*Triangle search*, in turn, takes advantage of Corollary 1 and identifies non-redundancy by only checking paths consisting of two edges. In other words, triangle search is based on 3-cliques search. Triangle search misses some redundant edges but is faster than global best path search.

*Iterative versus static pruning* The *iterative* variant works dynamically: changes to the graph take effect immediately and may affect the redundancy of other edges. Iterative pruning results in a completely pruned graph. The result is not unique but depends on the order in which edges are processed.

The *static* variant, in turn, determines the redundancy of all edges in a single batch, based on the original graph only. Then, all redundant edges are removed.

There is another important difference between the static and iterative variants. If the static method would be applied directly based on the definition (1), pruning would be too aggressive. In the case of ties, too many edges would be removed, affecting some best path qualities, possibly even leading to disconnected

components. Therefore an edge is removed only when the quality of the best path is strictly *higher* than the quality of the edge. This guarantees that all best path qualities are maintained. Static pruning thus prunes less edges than the iterative variant, and does not necessarily produce a non-redundant graph. The static method is faster, however, and its result is unique.

### 3.2   Outlines of algorithms

Of the four algorithmic variants that we present, the *Iterative-Global* variant is the most complete and the only one guaranteed to produce a completely pruned graph. It is outlined as Algorithm 1. It first converts a multigraph (with parallel edges) into a simple one (Line 2). Then it finds, for each edge, the best possible alternative path $P$ (Line 4). If it is at least as good as the edge, then the edge is pruned as redundant (Line 6).

---

**Algorithm 1** Iterative-Global Algorithm

---
**Input:** A weighted graph $G(V, E)$, a path quality function $q$
**Output:** Subgraph $H \subset G$
 1: $F \leftarrow E$
 2: If $F$ contains parallel edges, prune all but the best one.
 3: **for** each $e = \{u_i, u_j\} \in F$ **do**
 4:     Find path $P = \{\{u_i, u_{i+1}\}, \ldots, \{u_{j-1}, u_j\}\} \subset F \setminus \{e\}$ that maximizes $q(P)$
 5:     **if** $q(e) \leq q(P)$ **then**
 6:         $F \leftarrow F \setminus \{e\}$
 7: Return $H = (V, F)$

---

A straightforward implementation of the multigraph processing takes $O(|E|^2)$ time (Line 2). On Line 4, we use Dijkstra's algorithm with a complexity of $O\big((|E| + |V|) \log |V|\big)$ to find the best path. This is done for all $|E|$ edges. The computational complexity of the whole Iterative-Global method without any optimizations is then $O\big(|E|(|E| + |V|) \log |V|\big)$.

The *Iterative-Triangle* algorithm is very similar to the Iterative-Global one. The only difference is that on Line 4 of Algorithm 1 the best path search is replaced by the triangle search:

    4: Find path $P = \{\{u_i, u_k\}, \{u_k, u_j\}\} \subset F \setminus \{e\}$ that maximizes $q(P)$.

The computational cost of a triangle search for a single edge is $O(|V|)$. The total time complexity of Iterative-Triangle is thus $O(|E|^2 + |E||V|)$.

The *Static-Global* method, outlined in Algorithm 2, differs from Iterative-Global in two respects. First, all best paths are computed in a single batch in the set $E$ of original edges (Lines 3–4). Second, the path quality check on Line 6 requires proper inequivalence.

Our implementation computes all-pairs shortest paths in time $O(|V|(|E| + |V|) \log |V|)$. The total time complexity is thus $O(|E|^2 + |V|(|E| + |V|) \log |V|)$. This could be improved slightly by using Johnson's algorithm [4].

---

**Algorithm 2** Static-Global Algorithm

---

**Input:** A weighted graph $G(V, E)$, a path quality function $q$
**Output:** Subgraph $H \subset G$
 1: $F \leftarrow E$
 2: If $F$ contains parallel edges, prune all but the best one.
 3: **for** each $e = \{u_i, u_j\} \in F$ **do**
 4:     Find path $P = \{\{u_i, u_{i+1}\}, \ldots, \{u_{j-1}, u_j\}\} \subset E \setminus \{e\}$ that maximizes $q(P)$
 5: **for** each $e = \{u_i, u_j\} \in F$ **do**
 6:     **if** $q(e) < q(P)$ **then**
 7:         $F \leftarrow F \setminus \{e\}$
 8: Return $H = (V, F)$

---

The *Static-Triangle* variant is again very similar to the global one, and the difference is again only to replace best path search on Line 4 of Algorithm 2 with the triangle search:

4: Find path $P = \{\{u_i, u_k\}, \{u_k, u_j\}\} \subset E \setminus \{e\}$ that maximizes $q(P)$.

The *Static-Triangle* variant is the fastest and the least complete of the four variants. Its worst-case computational complexity is identical to *Iterative-Triangle*. However, as pruning takes place during the process, the complexity is typically lower.

## 4 Related work

Literature on network simplification and subgraph extraction is next. Here, we review some main approaches and their representatives.

*Simplification of flow networks* aims at removing edges while not affecting network flow for any node pair. This implies that the problem setting is more conservative than the one studied here. The methods proposed in [5, 6] are based solely on the topology of the graph, not the weights. Generalization of those methods to other kinds of graphs is a topic for possible further work.

*Pathfinder networks* [7, 8] are in many aspects similar to our pruned graphs and the global algorithm variants. The formulation is parameterized by $r$, to define Minkowski distance measure for pairs of nodes, and by $q$, a maximum path length considered (in other words, triangle search corresponds to $q = 2$ and global search to $q = |N| - 1$). Pathfinder networks then keep edges for which no better path exists with respect to $r$ and $q$. While originally proposed for psychological research, Pathfinder networks are now gaining popularity in visualization of very large co-citation networks.

*Neighborhood graphs* originate from computational geometry. Given a set of points in some space, the task is to link proximal points by edges. When exactly to link two points varies between methods. A popular instance is Relative Neighborhood Graph [9], where an edge connects $u, v \in V$ if and only if $d(u, v) \leq max(d(u, z), d(z, v))$. See Veltkamp [10] for an overview of geometric graphs.

The idea of relative neighborhood graphs can be applied to regular graphs—the distance between two nodes is the weight of the edge connecting them, or infinite if none exists. This is a special case of the framework proposed here.

*Minimum spanning tree* extraction is an old problem with specific solutions [11, 12]. The model proposed in this paper can be used to find spanning trees, among others, but not necessarily minimal ones.

## 5   Experiments

To assess the methods proposed in the previous sections, we conducted experiments with various real graphs from the biomedical domain. With the experiments we aim to find initial answers to the following questions. How many edges are removed? How do the removed edges affect more complex measures of relatedness? What are the removed edges like semantically? How efficient and scalable are the methods?

### 5.1   Experimental setup

Our data source is the Biomine database [1] which integrates information from twelve major biomedical databases. In the Biomine graph, nodes are biological entities such as genes, proteins, and biological processes. Edges correspond to known or predicted relations between entities, and have weights between 0 and 1.

For the tests, we picked 60 pairs of genes, such that each pair was related to the same phenotype according to Köhler et al. [13]. For each pair, the Biomine search engine[1] was then used to obtain a 500 node subgraph that contains the neighborhoods of the two given nodes. (Pairs whose neighborhoods were not connected were discarded.) For scalability tests, we additionally obtained a series of graphs for each pair, with sizes increasing from 500 to 3000 nodes.

We experimented with two classes of graphs, flow graphs and probabilistic ones, to get a perspective on the performance of the methods in different settings. In both cases, we directly used the weights provided by Biomine as the capacities or probabilities, respectively.

The algorithms were implemented in Java. Best paths were computed using a separate tool kindly provided by Lauri Eronen. All tests were run on standard PCs with $x86\_64$ architecture with Intel Core 2 Duo a 3.16GHz, running Linux.

### 5.2   Results

**5.2.1   How many edges are removed?** We first evaluate the effectiveness of the methods in terms of the number of edges removed. Given a transformation of a graph $G = (V, E)$ to $H = (V, F)$, we measure the relative size reduction $s = (|E| - |F|)/|E|$.

---

[1] biomine.cs.helsinki.fi

The results obtained for flow graphs are shown in Figure 3a. The Iterative-Global variant is the most effective, pruning even more than 30% of edges on average. Static-Global follows, pruning approximately 25%. The two Triangle variants have roughly equal performance, both pruning slightly less than 20%. Figure 3b shows the results obtained for probabilistic graphs. The fraction of edges removed by Global variants is around 9%, and for Triangle variants around 7%.



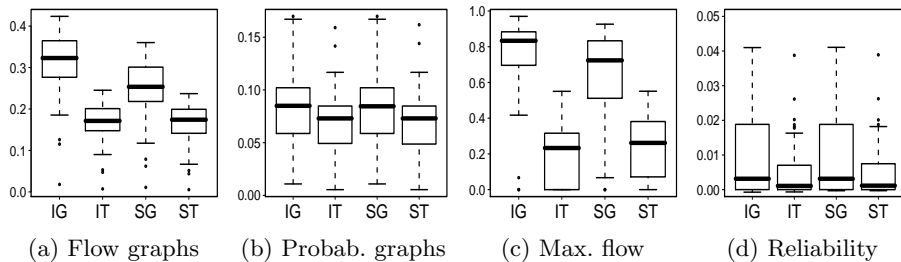(a) Flow graphs     (b) Probab. graphs     (c) Max. flow     (d) Reliability

Fig. 3: (a) and (b): Fraction of edges removed by each of the four algorithmic variants. (c) and (d): Fraction of more complex measures lost in simplification. Each boxplot shows the distribution of results over 60 test graphs. IG=Iterative-Global, IT=Iterative-Triangle, SG=Static-Global, ST=Static-Triangle

In absolute numbers, flow graphs lost approximately 120 to 240 edges out of 740 original ones. Probabilistic graphs lost approximately 50 to 60 edges. The results indicate that a relatively large number of edges can be removed without affecting the best path connectivity between *any* pair of nodes.

### 5.2.2   What are the effects on more complex connectivity measures?
In order to get a better understanding of the characteristics of the proposed methods, we next evaluate their effect on some other measures.

*Effect on maximum flow* For flow networks, a natural measure of connectivity between two given nodes is the maximum total flow from one node to the other, using all possible paths. In each of the graphs in our experiments, we also measured the maximum flow between the two nodes originally used to obtain the graph, computed with the Edmonds-Karp algorithm [14].

Figure 3c shows how much the maximum flow was reduced by each of the four algorithmic variants; $f = ((\mathrm{MaxFlow}(G) - \mathrm{MaxFlow}(H))/\mathrm{MaxFlow}(G)$. Iterative-Global loses approximate 80% of the flow capacity, closely followed by Static-Global. The Triangle variants only lose about 30% on average. (Iterative-Triangle seems to have lost slightly less flow capacity than Static-Triangle, but this difference is due to handling edges in random order.)

*Effect on graph reliability* In a probabilistic graph, the two-terminal network reliability (ttnr) is a natural measure of global connectivity between two given nodes [1]. The reliability can be estimated with a Monte Carlo approach: generate

a large number of realizations of the probabilistic graph, and count the relative frequency of graphs containing a path between the two given nodes. In our experiments, we used 10 million Monte Carlo realizations in the evaluation of each graph to minimize estimation error.

Figure 3d shows the relative loss of reliability, $r = (\text{ttnr}(G) - \text{ttnr}(H))/\text{ttnr}(G)$. All relative losses of Global variants are below 4%, and on average below 0.5%. Relative losses of Triangle variants are even less. In other words, edge pruning based on the best path quality had almost no effect on the more global network reliability.

**5.2.3    What are the removed edges like semantically?** An interesting issue is what the pruned edges are like. In Biomine, certain types of edges can be considered elementary; they connect entities that strongly belong together in biology. These include genes and the proteins they code for, homologous genes (in other words, the genes are essentially the same but in different organisms), and synonyms. An expert user most likely would not like any of these links to be pruned. On the other hand, since these relations are so elementary, other nodes need (and should) be connected to only one of them; the other connections could be considered semantically irrelevant.

With this idea, we designed the following experiment. We defined the edge types *codes for*, *is homologous to*, *subsumes*, and *has synonym* as "important." Then, we computed the number of those edges, adjacent to an important edge, that are "semantically irrelevant." For the sake of completeness, we also counted the number of "other edges" that are neither important nor adjacent to any important edge.



(a) Import.   (b) Irrelev.   (c) Other   (d) Parallel   (e) Running times
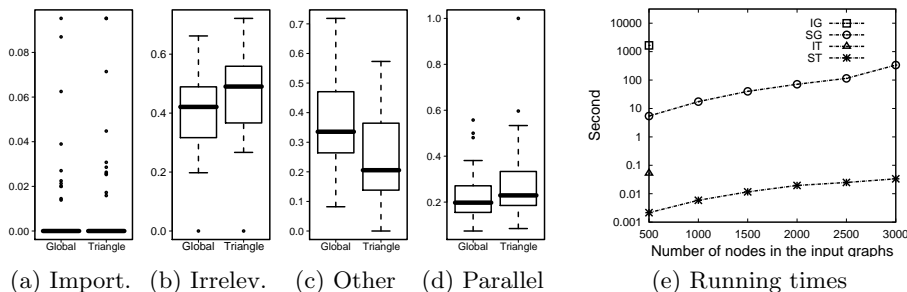
Fig. 4: (a)–(d): Shares of different semantic categories among all removed edges (static methods in probabilistic graphs; note different vertical scales). (e): Mean running times (in logarithmic scale) of 60 runs as functions of the size of the input flow network.

An analysis of the edges removed by the simplification method in probabilistic graphs shows that practically none of the removed edges were important (Figure 4a), 32–56% were irrelevant (b), and 12–48% were "other edges" (c), in other words those on which our crude semantic classification could not decide, and 15–33% were parallel ones (d). Looking at each of the semantic categories,

we observe that about 22–30% of semantically irrelevant edges were pruned, and 2–7% of other edges (results not shown). Even though the last percentage seems low, the category is by far the largest and, as stated above, stands for 12–48% of all removed edges. The results indicate that in our application the simplification method could considerably complement and extend expert-based or semantic methods, while not violating their principles.

**5.2.4  How efficient and scalable are the methods?** Finally, we compare running times of the four variants. For this purpose, we used differently sized variants from 500 to 3000 nodes for each of the 60 graphs.

The mean running times for flow graphs are given in Figure 4e (results are similar for probabilistic graphs). Static-Triangle always needed less than 0.1 second to complete, even for a graph consisting of 3000 nodes. (The other variants were considerably slower. However, this comparison is not fully fair and should be considered at most indicative: Since the best path computation was implemented by an external tool, the global versions and especially the iterative versions suffered from the unnecessary cost of repetitive passing of graph files and starting new processes.)

## 6  Conclusion

We have addressed the problem of network simplification. The viewpoint is path-oriented: In the simplified network, and for any pair of its nodes, the best path qualities have been maintained. The framework is applicable to a large variation of network types and path qualities, some of which we illustrated with examples. Example instances include relative neighborhood graphs, spanning trees, and certain Pathfinder graphs.

Based on properties of monotone path quality functions, we gave four algorithmic variants of varying computational complexity that simplify a given network with varying degrees of completeness. Experiments performed on 60 real biological networks illustrate the potential of this approach. A rough semantic analysis of the simplification indicates that in our experimental setting, practically no semantically important edges were removed. Instead, the proposed method could complement semantics-based simplification by identifying additional redundant edges.

The simplification methods are part of larger on-going, system-oriented projects (Bison[2] and Biomine[3]) aimed at intelligent analysis of networked information. One of the next steps is validation of the simplification methods in user studies. For future work, an interesting topic is further generalization of the framework to more greedy network compression where some loss of best path qualities could be traded against removal of more edges.

---

[2] www.bisonet.eu
[3] biomine.cs.helsinki.fi

# References

1. Sevon, P., Eronen, L., Hintsanen, P., Kulovesi, K., Toivonen, H.: Link discovery in graphs derived from biologican databases. In Leser, U., Naumann, F., Eckmann, B., eds.: 3rd International Workshop on Data Integration in the Life Sciences 2006 (DILS'06). Volume LNBI 4705. Springer-Verlag, Berlin/Heidelberg (2006) 35–49
2. Hintsanen, P., Toivonen, H.: Finding reliable subgraphs from large probabilistic graphs. Data Mining and Knowledge Discovery **17** (2008) 3–23
3. Berthold, M.R., Dill, F., Kötter, T., Thiel, K.: Supporting creativity: Towards associative discovery of new insights. In: Advances in Knowledge Discovery and Data Mining. Volume LNCS 5012. Springer, Berlin/Heidelberg (2008) 14–25
4. Johnson, D.B.: Efficient algorithms for shortest paths in sparse networks. Journal of the ACM **24**(1) (1977) 1–13
5. Biedl, T.C., Brejova, B., Vinar, T.: Simplifying flow networks. In: Mathematical Foundations of Computer Science 2000. Volume LNCS 1893. Springer-Verlag, Berlin/Heidelberg (2000) 192–201
6. Misiolek, E., Chen, D.Z.: Two flow network simplification algorithms. Information Processing Letters **97** (2006) 197–202
7. Schvaneveldt, R., Durso, F., Dearholt, D.: Network structures in proximity data. In: The Psychology of Learning and Motivation: Advances in Research and Theory. Volume 24. Academic Press, New York 249–284
8. Quirin, A., Cordon, O., Santamaria, J., Vargas-Quesada, B., Moya-Anegon, F.: A new variant of the pathfinder algorithm to generate large visual science maps in cubic time. Information Processing and Management **44** (2008) 1611–1623
9. Toussaint, G.T.: The relative neighbourhood graph of a finite planar set. Pattern Recognition **12**(4) (1980) 261–268
10. Veltkamp, R.C.: The gamma-neighborhood graph. Computational Geometry **1** (1991) 227–246
11. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society **7** (1956) 48–50
12. Osipov, V., Sanders, P., Singler, J.: The filter-kruskal minimum spanning tree algorithm. In Finocchi, I., Hershberger, J., eds.: ALENEX, SIAM (2009) 52–61
13. Köhler, S., Bauer, S., Horn, D., Robinson, P.: Walking the interactome for prioritization of candidate disease genes. American Journal of Human Genetics **82**(4) (April 2008) 949–958
14. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the Association for Computing Machinery **19**(2) (1972) 248–264