Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Master's Thesis Nr. 139

Systems Group, Department of Computer Science, ETH Zurich

Top-k Reliable Color Set in Uncertain Graphs

by

Andreas Nufer

Supervised by

Dr. Arijit Khan

11th September 2015

inf | Informatik
Computer Science

# TOP-K RELIABLE COLOR SET IN UNCERTAIN GRAPHS

Andreas Nufer

Master Thesis

Prof. Dr. Donald Kossmann
Dr. Arijit Khan

September 2015

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systems@ETH Zürich

# Abstract

In this thesis, we analyze the problem of finding the top-$k$ edge colors that maximize the reliability between a *set* of source nodes and a *set* of destination nodes in an uncertain, edge-colored graph.

Given more than one source and one destination, we further need to specify the maximization problem. Two types of maximization problems are studied. In the first, we maximize the pairwise reliability between nodes based on an aggregation function like Maximum, Average or Minimum. In the second, we maximize the connectivity between all nodes at once.

The top-$k$ color problem itself is NP-hard. We design effective, heuristic algorithms to solve these maximization problems in an efficient and reliable way. To carry this theoretical work on, we conduct an extensive empirical evaluation which shows that our solutions are scalable and highly accurate.

# Contents

# 1 Introduction

In graph theory, *uncertain graphs* are special graphs whose edges are accompanied with a probability of existence. They have a wide variety of applications in the real world such as noisy measurements, inference and prediction models, or explicit manipulation e.g for privacy purposes. A fundamental problem in uncertain graphs is *reliability query*, which asks to estimate the probability that a given destination node is reachable from a given source node. The reliability estimation problem has been widely studied in device networks [BALL86], social networks [JIN11], as well as in biological networks [SEVON06].

Most of these reliability queries over uncertain graphs are performed without considering any edge attributes, which we simply refer to as edge colors. Since complex networks, such as biological, social, and information networks usually exhibit diverse types of relationships among the entities, it is often meaningful to define reliability via a constrained set of edge colors [CHEN14].

Previous studies on the problem of *top-$k$ reliable color set uncertain graph* which maximizes reliability between a single source and a single destination node have been done by A. Khan *et al.* [KHAN15]. They proved its NP-hardness, that it is neither sub-modular nor super-modular and designed two baseline and a heuristic algorithm to efficiently solve the problem.

## 1.1 Our Contribution

In this thesis, we continue the work of A. Khan *et al*. We extend the problem for multiple source and destination nodes. With more than two nodes, we need to redefine the maximization problem. We study two types of maximization problems, one that considers pairwise reliability between source and destination nodes and one than considers the connectivity between all nodes at once. In the case of pair-wise reliability we use some aggregation functions like Average, Maximum or Minimum to maximize on different objectives. Then we develop a set of baseline and heuristic solutions for all of these maximization problems, based on the algorithms previously proposed. In addition to the study on multi source/destination problems,

we will also suggest some improvements to the original heuristic algorithm.

In the second part of the thesis, we evaluate the performance of our algorithms in a range of various experiments. We compare the execution time and the reliability of the found colors for both baselines and the heuristic solutions and we show that the heuristic approach provides good results in both. We run various experiments on different datasets and queries. We start by reproducing some of the results of A. Khan *et al* on single-source/destination queries and then continue with multiple sources and destinations. We also provide a detailed analysis on why and when the different algorithms perform better or worse.

# 2
# Background and Related Work

In this chapter we summarize the previous studies by A. Khan *et al.* on the problem of "*Top-k Reliable Edge Colors in Uncertain Graphs*". [KHAN15]

## 2.1 Preliminaries

**Problem formulation**
An uncertain, edge-colored, directed graph $\mathcal{G} = (V, E, C, P)$ is defined by the set of nodes $V$, the set of edges $E \subset V \times V$ and the set of edge colors $C$. The number of colors is designated as $n$, the number of edges as $m$. The function $C(e) \in C$ assigns a set of colors to an edge $e \in E$. The function $P : E \times C \longrightarrow (0, 1)$ assigns a conditional probability to an edge given a specific color, i.e., $P(e|c) \in (0, 1)$.

**Possible world semantics**
The bulk of the literature on uncertain graphs and device-network-reliability assumes the existence of an edge in the graph is independent of the existence of the other edges and interprets uncertain graphs according to the well-known possible-world semantics [FISH86, JIN11]. More precisely, given a predefined edge-color set $C_1$, a possible graph $G \subseteq \langle \mathcal{G}, C_1 \rangle$ is a pair $(V, E_G)$, where $E_G \in E$, and its sampling probability is:

$$Pr(G|C_1) = \prod_{e \in E_G} P(e|C_1) \prod_{e \in E \setminus E_G} (1 - P(e|C_1))$$

For a possible deterministic graph $G \subseteq \langle \mathcal{G}, C_1 \rangle$, we define an indicator function $I_G(s, t)$ to be 1 if there is a path in $G$ from a source node $s \in V$ to a target node $t \in V$, and 0 otherwise. Finally, the probability that $t$ is reachable from $s$ in the uncertain graph $G$ and via a predefined edge-color set $C_1$ is defined as the edge-color-constrained reliability from $s$ to $t$, and it is denoted by $R_{C_1}(s, t)$. The edge-color-constrained reliability is computed as follows.

$$R_{C_1}(s,t) = \sum_{G \sqsubseteq \langle \mathcal{G}, C_1 \rangle} [Pr(G|C_1) \times I_G(s,t)]$$

The number of possible worlds $G \subseteq \langle G, C_1 \rangle$ is exponential in the number of edges.

**Problem statement**
We are now ready to define our problem statement.

TOP-$k$ RELIABLE COLOR SET . *Given a source node $s \in V$ and a destination node $t \in V$ in an edge-colored, uncertain graph $\mathcal{G} = (V, E, C, P)$, and a small positive integer $k$, find the edge-color-set $C_1$ of size $k$ that maximizes the edge-color-constrained reliability $R_{C_1}$*

$$\begin{aligned} &\arg\max_{C_1 \subseteq C} R_{C_1}(s,t) \\ &\text{such that } |C_1| = k \end{aligned}$$

Intuitively, the top-$k$ reliable edge-colors create multiple paths of high probabilities from source to destination node.

**Hardness**
The top-$k$ color problem is NP-hard, which means it cannot be solved exactly in a reasonable amount of time for non-trivial cases. Unfortunately it is also neither sub-modular nor super-modular, which would allow us to efficiently find a (near-)optimal solution. We omit the proofs in this summary for the sake of simplicity.

## 2.2 Individual Top-k: Baseline 1

In the first baseline, the reliability for each color is estimated individually. This means, we calculate $R_{\{c\}}(s,t)$ for every $c \in L$ and select those $k$ colors with the highest reliability.

**Time complexity**
We estimate the reliability by applying a Monte Carlo (MC) sampling on the graph. To get a good estimate, we need to do $K$ iterations, resulting in a time complexity of $O(K(n+e))$ for MC. An estimate for all colors $|C|$ in the graph is needed. Selecting the top colors can be done in $O(k \log |C|)$. This results in a total complexity of $O(|C|K(n+e) + k \log |C|)$

**Difficulties**
If the source and destination nodes aren't connected by a single-colored path, the individual Top-$k$ approach won't find any colors.

For large graphs, the Monte Carlo sampling is very inefficient. Combined with a large $|C|$, the baseline becomes very slow.

## 2.3 Iterative Hill-Climbing: Baseline 2

We address the problem with paths of multiple colors in the second baseline, the *Iterative Hill-Climbing*. During the hill-climbing, we maintain a set of selected colors, starting with an empty set. In each iteration, we add a new color to our set: For each of the remaining colors, we estimate the reliability of the color in combination with the previously selected

colors. Then we include the color which improves the reliability the most. After $k$-iterations we find all top-$k$ colors.

**Time complexity**
Each iteration has a time complexity of $O(|C|K(n+e))$. We need a total of $k$ iterations, so the total complexity becomes $O(|C|kK(n+e))$.

**Difficulties**
Baseline 2 requires $k$ iterations over all colors, which is even slower than Baseline 1.

Baseline 2 partially solves the problem for multi-colored paths, as it may find those in subsequent iterations, it still cannot reliably detect the initial colors. Figure 2.1 illustrates this problem: in the first iteration, no single color has a better reliability from $s$ to $t$, none of them alone could complete the path. Only in the second iteration, after we would have randomly selected blue, red or black, Baseline 2 could predict the best color. But as long as the baseline randomly picks the wrong colors, it won't be able to find the a good color set. This is called the *"Cold start"* problem.

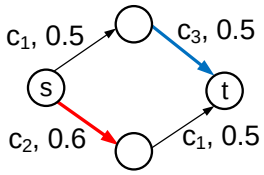



Figure 2.1: Multi-colored s-t path

## 2.4 Most-Reliable-Path based heuristic

With the Most-Reliable-Path based heuristic solution (Algorithm 2.1), A. Khan *et al.* address the cold-start problem and the possible scalability bottleneck of both baselines. The proposed heuristic algorithm consists of two steps.

**Most reliable path selection**
The idea behind the first step is to find a subset of the graph that is small and has a good reliability approximation for the connection between the source and the destination. Such a sub graph is induced by the most reliable paths. These paths can be found by any top-$k$ shortest path algorithm. The paper explicitly mentions Eppstein's algorithm, as it has the best time complexity. In order to use such an algorithm, the uncertain, multi-colored-edge graph has to be converted into a weighted graph with single-colored edges (Algorithm 2.2).
To build the weighted graph, every edge with $i$-colors is replaced by $i$ edges with a single color. In addition, every edge $e$ with color $c$ gets an assigned weight $W(e_i) = -\log P(e|c_i)$. The weight function ensures that the $n$th-shortest path in the weighted graph is equal to the $n$th-most reliable path in the uncertain graph.

**Iterative Path Inclusion**
In the second step, the shortest paths are iteratively included into a set of paths such that they maximize the reliability between the source and the destination without exceeding the color limit $k$. Unfortunately, the *Iterative Path Inclusion* problem is NP-hard; and it is neither sub-

**Algorithm 2.1** Most-Reliable-Path based heuristic

**Require:** Uncertain graph $\mathcal{G} = (V, E, C, P)$, source node $s \in V$ and destination node $t \in V$, number of top paths $r$, number of top colors $k$
**Ensure:** A color set $C_1$ that maximizes the edge-color-constrained reliability $R_{C_1}$
1: Convert the uncertain graph $\mathcal{G}$ into an weighted graph $\mathcal{G}'$ (2.2)
2: Find the top-$r$ shortest paths $\mathcal{P}$ from $s$ to $t$ on $\mathcal{G}'$ using Eppstein's algorithm.
3: Use the Iterative Path Inclusion algorithm (2.3) to obtain the top-$k$ color set $C_1$ from $\mathcal{P}$.
4: Add random colors from $C$ to $C_1$ until $|C_1| = k$.
5: **return** $C_1$

**Algorithm 2.2** Convert unreliable graph to weighted graph

**Require:** Uncertain graph $\mathcal{G} = (V, E, C, P)$, source node $s \in V$ and destination node $t \in V$
**Ensure:** Weighted graph with single-colored edges $|C(e)| = 1, \forall e \in E$
1: **for all** $e \in E$ **do**
2: let $C(e) = \{c_1, c_2, ..., c_i\}$
3: replace $e$ by $i$ edges $\{e_1, e_2, ..., e_i\}$
4: assign edge-weight $W(e_i) = -\log P(e|c_i)$
5: **end for**
6: **return** the constructed multigraph $\mathcal{G}' = (V, E', C, W)$

modular, nor super-modular with respect to the inclusion of paths. To overcome this problem, A. Khan *et al.* propose a greedy heuristic approximation (Algorithm 2.3). They start with an empty set of included paths $\mathcal{P}_1$. In each iteration, they select the path $P^*$ that will increase the reliability the most when added to $\mathcal{P}_1$, such that the total number of different colors in $\mathcal{P}_1$ does not exceed $k$. To estimate the reliability, they again use a Monte Carlo sampling with $K$ samples. The iteration continues as long as there are any paths left that are neither included nor violate the color constraint. When the *Iterative Path Inclusion* finds less than $k$ colors, the remaining colors are selected randomly from the available colors in $C$.

**Time complexity**
Define $n'$ and $e'$ as the number of nodes and edges of the sub graph $\mathcal{G}'$ induced by the top-$r$ paths between $s$ and $t$. In each iteration we sample at most $r$ paths, with at most $r$ iterations. Each MC-sampling with $K$ samples has a time complexity of $O((n' + e')K)$. Total time complexity for the *Iterative Path Inclusion* is therefore $O(r^2(n' + e')K)$. As the sub graph $\mathcal{G}'$ is much smaller than $\mathcal{G}$, the heuristic approach is much faster than the baselines.

**Algorithm 2.3** Iterative Path Inclusion

**Require:** Uncertain graph $\mathcal{G}_\mathcal{P}$ induced by the top-$r$ path set $\mathcal{P}$ between $s$ to $t$, color limit $k$
**Ensure:** A subset of paths $\mathcal{P}_1 \subseteq \mathcal{P}$ that maximizes $Rel_{\mathcal{P}_1}(s, t)$, while $|C(\mathcal{P}_1)| \leq k$
1: $\mathcal{P}_1 = \emptyset$
2: **while** $|C(\mathcal{P}_1)| < k$ **do**
3: $P^* = \arg\max_{P \in \mathcal{P} \setminus \mathcal{P}_1} Rel_{\mathcal{P}_1 \cup \{P\}}(s, t)$, such that $|C(\mathcal{P}_1 \cup \{P\})| \leq k$
4: $\mathcal{P}_1 = \mathcal{P}_1 \cup \{P^*\}$
5: **end while**
6: output $C(\mathcal{P}_1)$

# 3

# Single-Source/Destination Heuristic

Before we discuss the heuristic algorithms for multiple-source/destination problems, we review the proposed *Most-Reliable-Path based Heuristic* solution. We suggest some new modifications to the existing algorithms and present an alternative solution to find the top-$k$ shortest paths. These modifications will also impact the performance of the multiple-source/destination algorithms.

## 3.1 Shortest paths

A centerpiece of the heuristic algorithm is to find the shortest paths between the source and the destination. A. Khan *et al.* suggest Eppstein's algorithm [EPP98], which has best time complexity for the problem.

### 3.1.1 Eppstein and Lazy Eppstein

Eppstein's algorithm finds the $r$-shortest paths between two nodes in $O(m + n \log n + r)$ time. The algorithm itself is rather complicated so we omit the details here. In simple terms, Eppstein first calculates the shortest paths to the destination for each node in the graph. Then it constructs a new graph $D(G)$ to keep track of all the possible alternatives routes derived from the shortest path. In its basic version, it takes $O(m+n \log n)$ time to build $D(G)$, which consumes the most part of the algorithms execution time. In practice, often only a few nodes and edges of $D(G)$ are used to find the shortest paths. Therefore, building the full graph in advance is an unnecessary waste of resources in most cases.
Vıctor M. Jimenez and Andres Marzal [JIME03] propose a lazy version of Eppstein's algorithm, where only those parts of $D(G)$ are computed that are actually used. However, even the lazy version requires to calculate the shortest path for each node to the destination node. This will take up $O(m+n \log n)$ time and we will see later that the additional time makes the algorithm less competitive in comparison to the baselines. We address this issue by proposing a modified version of the Lazy Eppstein's algorithm:

### 3.1.2 Lazier Eppstein's algorithm (H-LE)

Knowing the shortest path to the destination $t$ for all nodes in the graph is often not needed. Lazier Eppstein uses the same concept of lazy evaluation as Lazy Eppstein does and applies it to the shortest paths calculation. Our modification affects two of the algorithm's sub-procedures:

- $BuildTree()$: Builds the tree of shortest paths from all nodes to the destination.
- $GetShortestPath(v)$: Returns the shortest path from $v$ to the destination.

In our modified version, we introduce an additional parameter to $BuildTree()$ - the stop node. While constructing the tree of shortest paths, we set a flag on every node after having found its shortest path and we pause the construction as soon as we flag the stop node. For the initial call, the stop node will be the source to get the shortest path from the source to the destination. In $GetShortestPath(v)$ we add a check if the shortest path for $v$ has already been found. If not, we resume constructing the shortest path tree with $v$ as stop node. $GetShortestPath(v)$ is needed to find alternative paths along the neighboring nodes of the shortest path.

Under certain conditions, such as in large graphs with close source/destination pairs, the performance boost can be significant. However, in the worst case, Lazier Eppstein performs equal to Lazy Eppstein and the original version.
Unfortunately, Lazier Eppstein can only be applied on undirected graphs. In directed graphs, there might be nodes that can be reached from the source node, but which themselves do not have a shortest path to the destination. This cases cannot be identified without computing the whole tree of shortest paths. In undirected graphs, this only happens when there is no connection between the source and the destination.

**Difficulties**
Eppstein may find paths which have cycles. A cycled path visits some nodes and edges multiple times. Having path with cycles is not helpful for our purpose. When there are paths that only differ in the number of repeated cycles, we do not get more color information than we'd get from just one path. The alternative is to use a shortest path algorithm that doesn't allow cycles. The best available algorithm with these constraints is Yen's algorithm [YEN71], but it has a time complexity of $O(rnm + rn^2 \log n)$ which is considerably worse than Eppstein.

In the remaining document, the Lazier Eppstein's algorithm will be referred as H-LE.

### 3.1.3 Modified General Dijkstra (H-MGD)

As an alternative to Eppstein we introduce a version of the General Dijkstra algorithm (3.1 [1]). General Dijkstra is the generalized form the the Dijkstra shortest-path algorithm. Its time complexity is much worse that of Eppstein, so in the general case, Eppstein is by far the better choice. The advantages of General Dijkstra are its simplicity, and the fact that it expands around the source node (locality), whereas Eppstein might need to maintain structures over the whole graph. In the special case where the source/destination nodes are close to each other, it might be as fast as Eppstein.

To keep the execution time competitive with Baseline 1, we add a limit on how many nodes are expanded (Line 6 in 3.1). The consequence is, that in some cases none or not all shortest paths will be found. To lessen this effect, we will accept a "shortest path" as soon as we reach the destination through an edge from a neighbor (Line 14-16), instead of when we pop the destination as the smallest element from the priority queue $B$.

**Difficulties**
Due to the modification, the algorithm will no be able to find all existing top-$r$-shortest-path. And some of the paths found are not actually the shortest paths. However, these false-shortest-path will have less hops than their counter parts which might be an advantage when the color limitation $k$ is tight.

In the remaining document, the modified General Dijkstra algorithm will be referred as H-MGD.

## 3.2 Iterative Path Inclusion

In the *Iterative Path Inclusion* algorithm, as proposed by A. Khan *et al.*, paths are iteratively added to the selected-paths set by including the best remaining path that meets the color constraint. To determine the best path, an Monte-Carlo-sampling estimation is performed.
Given that a MC-sampling is costly, it is beneficial to reduce the number of estimates. Obviously, we don't need to test the paths that fail the color constraint. In addition, we can also skip paths which wouldn't add new colors to $\mathcal{P}$ by simply adding them to $\mathcal{P}$ without estimation. This may change the order how paths are selected, but it does not include new paths as we know they would be eventually added anyway as they cannot violate the color constraint. Changing the inclusion order might even improve the quality as subsequent estimations are based on more data (paths).

Algorithm 3.2 shows the *Modified Iterative Path Inclusion* (Lines 3-7). The time complexity does not change for the worst case, but we expect to get a better average run time performance.

[1] Source: https://en.wikipedia.org/wiki/K_shortest_path_routing, 2005-08-10

**Algorithm 3.1** Modified General Dijkstra $r$-shortest paths (Source: Wikipedia)

**Require:** Uncertain graph $\mathcal{G} = (V, E, C, P)$, source and destination node $s, t \subseteq V$, the number of shortest paths $r$.

**Ensure:** The set $P$ containing top-$r$ shortest paths from $s$ to $t$.

let $B$ be an empty priority queue.
$loops = 0$
$P = \emptyset$
$count_u$ = 0, for all $u$ in $V$
insert path $P_s$ = $s$ into $B$ with cost 0
**while** $B \neq \emptyset$ and $count_t < r$ and $loops <$ LoopLimit **do**
  let $P_u$ be the shortest cost path in $B$ with cost $W$
  $B = B - \{P_u\}$
  $count_u = count_u + 1$
  **if** $count_u \leq r$ **then**
    **for** vertex $v$ adjacent to $u$ **do**
      let $P_v$ be a new path with cost $W + w(u, v)$ formed by concatenating edge $(u, v)$ to path $P_u$
      insert $P_v$ into $B$
      **if** $v = t$ **then**
        $P = P \cup P_u$
      **end if**
    **end for**
  **end if**
  $loops = loops + 1$
**end while**
**return** $P$

**Algorithm 3.2** Modified Iterative Path Selection Algorithm

**Require:** Uncertain graph $\mathcal{G}_\mathcal{P}$ induced by the top-$r$ most-reliable path set $\mathcal{P}$ between $s$ to $t$, the number of colors $k$

**Ensure:** A subset of paths $\mathcal{P}_1 \subseteq \mathcal{P}$ that maximizes $Rel_{\mathcal{P}_1}(s, t)$, while $|C(\mathcal{P}_1)| \leq k$

$\mathcal{P}_1 = \emptyset$
**while** $|C(\mathcal{P}_1)| < k$ **do**
  **for** $P$ in $\mathcal{P} \setminus \mathcal{P}_1$ **do**
    **if** $C(P) \subseteq C(\mathcal{P}_1)$ **then**
      $\mathcal{P}_1 = \mathcal{P}_1 \cup \{P\}$
    **end if**
  **end for**
  $P^* = \arg\max_{P \in \mathcal{P} \setminus \mathcal{P}_1} Rel_{\mathcal{P}_1 \cup \{P\}}(s, t)$, such that $|C(\mathcal{P}_1 \cup \{P\})| \leq k$
  $\mathcal{P}_1 = \mathcal{P}_1 \cup \{P^*\}$
**end while**
output $C(\mathcal{P}_1)$

# 4

# Multi-Source/Destination Heuristic

In this chapter we discuss the main topic of this thesis, the top-$k$ color problem with multiple source/destination nodes. In order to design a solution, we need to specify the maximization problems. We find two fundamental types of maximizations:

In the first type, we look at the pairwise connections between all source/destination nodes $\langle s, t\rangle \in S \times T$. These pair-connections can be combined by using an aggregation function. In this thesis we cover three of the most common aggregations:

- *Max-Average*: Find the top-$k$ colors such that the average reliability over all $\langle s, t\rangle$-pairs is maximized. This is equivalent to the maximization of the sum over all $\langle s, t\rangle$-pairs.
- *Max-Maximum*: Find the top-$k$ colors to maximize the reliability over the $\langle s, t\rangle$-pair with the highest reachability.
- *Max-Minimum*: Find the top-$k$ colors such that the reliability of the $\langle s, t\rangle$-pair with the lowest reachability is maximized.

In the second type, we consider all nodes as equivalent peers. Similarly to the single source case, one can find the top-$k$ colors such that the connectivity between all peers is the most reliable. We refer to this as the *Max-Connectivity* problem.

## 4.1 Max-Average

When we consider a set of destination nodes that we want to reach from a set of source nodes, we are often not interested in the reliability of a particular pair of source/destination nodes. We rather want a good reliability over all node pairs $\langle s, t\rangle \in S \times T$. This problem is equivalent to the Max-Average maximization problem.

**Application**
*Social networks:* The Max-Average problem occurs naturally in social networks like Twitter when vendors or service providers want to reach new potential customers (vertices) through their existing followers via re-tweets (edges). The probability that followers re-tweet is based on their interest in a specific topic/hashtag (colors). So vendors want to know which topics they should promote to maximize the reliability to reach their target group. They are interested in a maximal reachability to their target group rather than in the reachability of a single target.

**Problem formalization**
We formally note the Max-Average maximization problem in a graph $\mathcal{G} = (V, E, C, P)$ as follows:

$$\underset{C_1 \subseteq C}{\arg\max} \sum_{\langle s,t\rangle \in S \times T} R_{C_1}(s, t)$$
$$\text{such that } |C_1| = k$$

where $R_{C_1}(s, t)$ returns the probability of a connection between $s$ and $t$ while only considering edges with colors in $C_1$. $S \subseteq V$ is the set of source nodes, $T \subseteq V$ is the set of destination nodes and $C_1 \in C$ is the set of selected colors that solves Max-Average top-$k$ colors problem.

### 4.1.1 Heuristic

The idea behind the heuristic solution for the Max-Average problem is to iteratively include the best paths from all pairs simultaneously. We create two new nodes $s'$ and $t'$. Then we find the top-$r$ shortest paths for each $\langle s, t\rangle$-pair, connect the first node of each path with $s'$ and the last node with $t'$ as illustrated in Figure 4.1. Finally, we use the *Iterative Path Inclusion* algorithm over $s'$ and $t'$ to find the top colors.

**Special case with $S \cap T = \emptyset$**
We start by introducing a solution with the restriction that the source and destination nodes are disjointed ($S \cap T = \emptyset$). In our new algorithm, we reuse the base algorithms for the single-source/destination case. First, we convert the uncertain graph into a weighted multigraph with single-colored edges (Algorithm 2.2). Then we add two new nodes $s'$ and $t'$ to the graph (Figure 4.1). Next, we find the top-$r$ shortest paths for each $\langle s, t\rangle$-pair. We prepend each path with an new edge from $s'$ to $s$ and append an other edge from $t$ to $t'$. The new edges have a constant weight (typically 0) and can reuse a color that is already present in the path. We combine all paths into a single set $\mathcal{P}'$ and use *Iterative Path Selection* to find the top-$k$ colors for $s'$ and $t'$ which solves the Max-Average problem (Algorithm 4.1).

**General case with $S \cap T \subseteq V$**
If $S \cap T \neq \emptyset$ there is a set of nodes such that there is an path from $s'$ to $t'$ in the form of $s' \xrightarrow{c=0} v \xrightarrow{c=0} t'$ with $v \in S \cap T$. Such paths will negatively impact the path selection an must be removed beforehand. They are not needed for the reachability estimation as their reliability is constant regardless the selected colors.

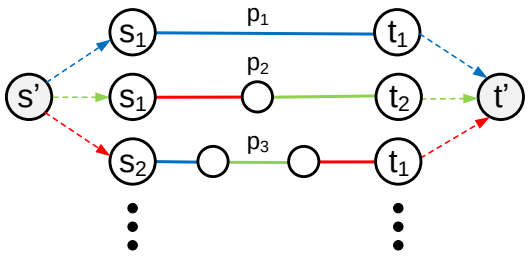



Figure 4.1: Extended top-$r$ paths for Max-Average

**Time complexity**
*Construction complexity:* The cost to add the new nodes is constant. The cost to extend the paths is $O(|S| + |T|)$.

*Total complexity:* Time to find the shortest paths for each pair is $O((|S| \cdot |T|)(m + n \log n + r))$. The time to select the top-k colors from the $(|S| \cdot |T|)r$ paths is $O((|S| \cdot |T|r)^2(n' + e')K)$ where $n'$ and $e'$ are the number of nodes and edges of the graph induced by the shortest paths and $K$ is the number of samples used in each Monte-Carlo sampling.

### 4.1.2 Baselines

In order to compare the heuristic solution to the baselines, we need to adapt them to the Max-Average problem.

**Baseline 1**
Baseline 1 for Max-Average estimates the colors individually using the sum of estimations over all $\langle s, t \rangle$-pairs. This results in a time complexity of $O(|C|K(n + e)(|S| \cdot |T|) + k \log(|C|))$.

**Baseline 2**
The modifications in Baseline 2 are similar to Baseline 1. Each color in each iteration is estimated over the sum, with a total complexity of $O(|C|kK(n + e)(|S| \cdot |T|))$.

**Algorithm 4.1** Path-based heuristic for Max-Average

**Require:** Uncertain graph $\mathcal{G} = (V, E, C, P)$, a set of source nodes $S \subseteq V$, a set of destination nodes $T \subseteq V$, number of top paths $r$, number of top colors $k$

**Ensure:** A subset of colors $C_k$ that maximizes the average reachability over all $\langle s, t\rangle \in S \times T$

```
convert probabilities to weights
P = ∅
for ⟨s, t⟩ in S × T do
    add top-r shortest path from s to t into P'
end for
for p in P' do
    prepend edge e_s = (s', s) to p with W(e_s) = 0 and C(e_s) ∈ C(p)
    append edge e_t = (t, t') to p with W(e_t) = 0 and C(e_t) ∈ C(p)
end for
find top-k colors C_k on P' using Iterative Path Inclusion
return C_k
```

## 4.2 Max-Maximum

Another interesting scenario is to find the top colors of the most connected node pair. The Max-Maximum maximization problem finds the top-$k$ colors of the particular $\langle s, t\rangle$-pair which has the highest reachability of all pairs with respect to their individual top-$k$ colors.

**Application**
*Social networks:* Sales promoter often use social platforms to sale their product. Often a promoter and its target are active on multiple social networks like Twitter, Facebook, Google+ etc (source and destination nodes). Naturally, the promoter is interested in maximizing the reachability to his target regardless the platform, what corresponds to the Max-Maximum problem.

**Problem formalization**
We formally note the Max-Maximum maximization problem in a graph $\mathcal{G} = (V, E, C, P)$ as follows:

$$\arg\max_{C_1 \subseteq C} \max_{\langle s,t\rangle \in S \times T \setminus \{\langle v,v\rangle | v \in S \cap T\}} R_{C_1}(s, t)$$
$$\text{such that } |C_1| = k$$

where $R_{C_1}(s, t)$ returns the reliability between $s$ and $t$ while considering only edges with colors in $C_1$. $S \subseteq V$ is the set of source nodes, $T \subseteq V$ is the set of destination nodes and $C_1 \in C$ is the set of selected colors that solves the Max-Maximum top-$k$ colors problem.

### 4.2.1 Heuristic

Since the top-$k$ colors for each $\langle s, t\rangle$-pair might be different, there is no way to to process all pairs at once without losing accuracy. So, the most accurate solution is to individually calculate the top-$k$ color set for each $\langle s, t\rangle$-pair and to choose the one with the highest reliability.

**Special case with $S \cap T = \emptyset$**
To find the top-$k$ colors for a pair, we can reuse the heuristic algorithm for single-source/destination. Then we estimate the reachability for each pair to determine the maximal value. Since the *Iterative Path Inclusion* algorithm already uses MC-sampling to evaluate the optimal paths, we can reuse these values to improve the performance.

**General case with $S \cap T \subseteq V$**
When a node $u$ is both in $S$ and in $T$, the above solution will return an arbitrary set of colors as $R_{C_1}(u, u) = 1$ and will always be considered as the optimal solution. If this behavior is not requested, it is sufficient to ignore all pairs of type $\langle v, v \rangle$ where $v \in S \cap T$ (Algorithm 4.2).

**Algorithm 4.2** Path-based heuristic for Max-Maximum

**Require:** Uncertain graph $\mathcal{G} = (V, E, C, P)$, a set of source nodes $S \subseteq V$, a set of destination nodes $T \subseteq V$, number of top paths $r$, number of top colors $k$
**Ensure:** A set of $k$-colors such that the maximal reachability between any pair of $s$ and $t$ is maximized.
1: get set of ordered pairs $Q = S \times T$
2: remove pairs of the same node from $Q = Q \setminus \langle u, u \rangle, u \in V$
3: convert $\mathcal{G}$ to a weighted multigraph $\mathcal{G}'$ using Algorithm 2.2
4: initialize $p_{top} = 0, C_{top} = \emptyset$
5: **for** $\langle s, t \rangle$ in $Q$ **do**
6: calculate a set $\mathcal{P}$ of r-shortest paths from $s$ to $t$
7: calculate top $k$-colors $C$ and reliability $p$ for $\mathcal{P}$ using Algorithm 2.3.
8: **if** $p_{top} \leq p$ **then**
9: set $p_{top}$ to $p$ and $C_{top}$ to $C$
10: **end if**
11: **end for**
12: **return** $C_{top}$

**Time complexity**
For each pair we have to find the top colors. The selection of the maximal color set is $O(|S| \cdot |T|)$. So the total time complexity is $O((|S| \cdot |T|)((m + n \log n + r) + (r^2(n' + e')K)))$ where $n'$ and $e'$ are the number of nodes and edges of the graph induced by the shortest paths and $K$ is the number of samples used in each Monte-Carlo sampling and $r$ is the number of shortest paths.

### 4.2.2 Baselines

The baseline adaptions for the Max-Maximum problem are the same as for the heuristic solution.

**Baseline 1**
Find the top-$k$ colors for each $\langle s, t \rangle$-pair using the single-Baseline 1. Then estimate the $\langle s, t \rangle$-reachability. Finally, select the color set with the highest estimate. Time complexity for Baseline 1 is $O((|S| \cdot |T|)(|C|K(n + e) + k \log |C|))$.

**Baseline 2**
Same principle as Baseline 1 but with using single-Baseline 2 to find the top-$k$ colors. Time complexity for Baseline 2 is $O((|S| \cdot |T|)(|C|kK(n+e)))$.

## 4.3 Max-Minimum

The previously described aggregations are biased towards strong connected pairs. To avoid this bias we focus on optimizing the weakly connected pairs. This corresponds to the Max-Minimum problem in which we maximize the reliability of the least connected pair.

**Application**
*Marketing:* Nowadays the success of a blockbuster movie heavily depends on mouth-to-mouth-marketing. Recommendations (edges) among friends (vertices) are influenced by the features (colors) of a movie. Movie marketing is interested in maximizing its audience by promoting the right features. However, only a small percentage of the reached people are going to watch the film, thus the movie studios want to optimize the minimum reachability to all of its potential audience.

**Problem formalization**
We formally note the Max-Minimum maximization problem in a graph $\mathcal{G} = (V, E, C, P)$ as follows:

$$\arg\max_{C_1 \subseteq C} \min_{\langle s,t \rangle \in S \times T} R_{C_1}(s,t)$$
$$\text{such that } |C_1| = k$$

where $R_{C_1}(s,t)$ returns the reliability between $s$ and $t$ while considering only edges with colors in $C_1$. $S \subseteq V$ is the set of source nodes, $T \subseteq V$ is the set of destination nodes and $C_1 \in C$ is the set of selected colors that solves Max-Minimum top-$k$ colors problem.

**Difficulties**
Baseline 2 is particularly prone to the "cold start" problem for solving the Max-Minimum maximization. Looking at each $\langle s,t \rangle$-pair individually does not work, because a good color set for one pair is not necessarily good for another. A strongly connected pair will have no positive effect on the overall outcome if there are unconnected pairs left.

### 4.3.1 Heuristic

We begin by collecting the shortest paths for each $\langle s,t \rangle$-pair. However, applying the Iterative Path Inclusion algorithm to the shortest path set becomes a little bit tricky. Previously, we either put the paths from all pairs into a single set, or we handled the pairs individually. This does not suit well in the Max-Minimum case as we would neglect less connected $\langle s,t \rangle$-pairs. We can use the *modified Iterative Path Inclusion* (Algorithm 4.5) that addresses the issue, but this is not enough. Iterative Path Inclusion aims for a short path length (lower path cost), but a short path might not have necessary less hops than a longer path, and more hops usually means more colors. When we have many pairs and a limited budget of colors, wasting too many colors on a single pair connection might prevent us from finding a path for another pair.

To fix this issue, we add an additional step: *Find Minimum Color Set*. This is similar to the iterative path inclusion problem, except we include the paths by number of colors and not by probability.

**General case with $S \cap T \subseteq V$**
Our heuristic solution has three phases (Algorithm 4.3). In the *first* phase, we convert the uncertain graph into an a weighted multigraph $\mathcal{G}'$ and find the top-$r$ most reliable paths for each $\langle s,t\rangle$-pair. We now have $|S| \cdot |T|$ sets of top-$r$ paths. In the *second* phase, we want to find a color set $C_b \subseteq C$ that covers the colors of at least one path of each $\langle s,t\rangle$-pair. (Find Minimum Color Set). In the *third* phase we find more colors using *Iterative Path Inclusion for Max-Minimum*.

**Find Minimum Color Set**
We formally define our minimum color set problem as follows:

MINIMUM COLOR SET. *Given a set of node-pairs $Q = S \times T$, a set of paths $\mathcal{P}$, an uncertain graph $\mathcal{G} = (V, E, C, P)$ induced by $\mathcal{P}$, and a small positive integer $k$, find the smallest edge-color-set $C_1$ of a size not larger than $k$ such that the edge-color-constrained reliability $R_{C_b}$ for each pair in $Q$ is larger than zero.*

$$\arg\min_{C_b \subseteq C} |C_b|$$
$$\text{such that } \forall \langle s,t\rangle \in Q : R_{C_b}(s,t) > 0$$

Unfortunately the Minimum Color Set problem is NP-hard, so we can only find an approximate result. We design a heuristic solution to find such an approximate result in 4 steps (Algorithm 4.4):

> *Step 1:* Collect all paths from $\mathcal{P}$ that belong to a $\langle s,t\rangle$-pair which is not yet flagged as connected. Select the path that will add the fewest new colors to the color set $C_b$.
>
> *Step 2:* Add the colors from the selected path to $C_b$ and flag its corresponding $\langle s,t\rangle$-pair as connected.
>
> *Step 3:* If the number of colors in $C_b$ exceeds $k$, try to remove a color from $C_b$ such that the already connected pairs remain connected. If this is not possible, then abort and return a set of random colors.
>
> *Step 4:* As long as there are unconnected pairs, go back to *Step 1*.

*Time Complexity:* Let $i = |S \times T|$ be the number of node pairs and $r' = ir$ the number of top-$r$ paths over all pairs. In each iteration, we find the path with the least new colors in $O(r'k \log k)$, and remove excessive colors in $O(r'k^2 \log k)$. We have $i$ iterations, thus the total complexity is $O(ir'k^2 \log k)$.

**Iterative Path Inclusion for Max-Minimum**
Once we have found a color set that ensures a connection between each $\langle s,t\rangle$-pair, we can use *Iterative Path Inclusion* to fill up to remaining colors such that we maximize the minimal reachability.

The *Iterative Path Inclusion for Max-Minimum* is a generalization of the *Iterative Path Inclusion* for the single-source case with an optimization towards weakly connected pairs. We formally define the problem as:

ITERATIVE PATH INCLUSION FOR MAX-MINIMUM. *Given a set $\mathcal{P}$ of the top-$r$ most reliable paths for each $\langle s,t\rangle \in S \times T$, find the subset $\mathcal{P}_1 \subset \mathcal{P}$, such that the lowest reliability $Rel_{\mathcal{P}_1}(s,t)$ of any $\langle s,t\rangle \in S \times T$ is maximized; while the total number of colors on the paths in $\mathcal{P}_1$ does not exceed $k$.*

$$\arg\max_{\mathcal{P}_1 \subseteq \mathcal{P}} \min_{\langle s,t\rangle \in S\times T} Rel_{\mathcal{P}_1}(s,t)$$
$$\text{such that } |\bigcup_{e\in\mathcal{P}_1} C(e)| \leq k$$

As a generalization of the *Iterative Path Inclusion* from Section 2.4, the problem is NP-hard.

**Theorem 1.** *The Iterative Path Inclusion for the Max-Minimum problem is neither sub-modular nor super-modular with respect to inclusion of paths.*

*Proof.* A function $f$ is sub-modular if it satisfies the constraint $f(A \cup x) - f(A) \geq f(B \cup x) - f(B)$ for all sets $A \subseteq B$ and all elements $x$. Figure 4.2 shows a counter example to this constraint.
Let $A = e_1e_3$, $B = e_1e_2e_3$, $x = e_4$ and $f(P) = \min_G Rel(P)$.

$f(A) = 0.1$, $f(A \cup x) = 0.1$
$f(B) = 0.5$, $f(B \cup x) = 0.6$
$f(A \cup x) - f(A) = 0 < 0.1 = f(B \cup x) - f(B)$

Therefore $f$ is not sub-modular.

A function $f$ is super-modular if it satisfies the constraint $f(A \cup x) - f(A) \leq f(B \cup x) - f(B)$ for all sets $A \subseteq B$ and all elements $x$. Figure 4.3 shows a counter example to this constraint.
Let $A = e_1e_3$, $B = e_1e_2e_3$, $x = e_4$ and $f(P) = \min_G Rel(P)$.

$f(A) = 0.1$, $f(A \cup x) = 0.5$
$f(B) = 0.2$, $f(B \cup x) = 0.5$
$f(A \cup x) - f(A) = 0.4 > 0.3 = f(B \cup x) - f(B)$

Therefore $f$ is not super-modular. □

We design a heuristic solution, based on the *Iterative Path Inclusion* algorithm, which includes the paths with respect to the least reachable $\langle s,t\rangle$-pair. We define a couple of functions: $pathset(P)$ returns the corresponding path set for the given path $P$, $\mathcal{P}(G)$ returns a path set that induces the given graph, the function $Rcol_G(s,t,C_1)$ returns the reliability between $s$ and $t$ in graph $G$ using only edges that have a color in $C_1$, and the function $C(\mathcal{P})$ returns the distinct edge-colors in the set of paths $\mathcal{P}$.

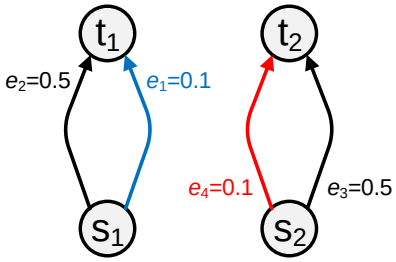



Figure 4.2: Non-Sub-Modularity Example

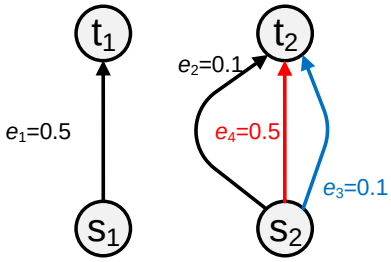



Figure 4.3: Non-Super-Modularity Example

We begin each iteration by finding the $\langle s, t\rangle$-pair with the lowest reachability on its associated graph $\mathcal{G}_{\mathcal{P}}$ with respect to the selected colors $C$. From the top-$r$ path set of this pair, we then include the path that increases its reliability the most. As long as we can choose more colors, we find a next pair and repeat (Algorithm 4.5). As long as each pair has a different reachability, the algorithm will guarantee that the minimum reachability over all pairs is increased in each iteration.

*Time Complexity:* Define $n_j < n'$ and $e_j < e'$ as the number of nodes and edges of the graph $\mathcal{G}_j$ induced by the top-$r$ paths for $\langle s_j, t_j\rangle$ in $S \times T$, $i = |S \times T|$ as the number of node pairs, $r' = ir$ as the total number of paths. In each inclusion we use MC-sampling to find the least connected pair in $O(i(n' + e')K)$, and the best path in $\mathcal{G}^*$ in $O(r(n' + e')K)$. We have to include at most $r'$ paths, resulting in a total time complexity of $O(r'(r + i)(n' + e')K)$.

**Algorithm 4.3** Path-based heuristic for Max-Minimum

**Require:** Uncertain graph $\mathcal{G} = (V, E, C, P)$, a set of source nodes $S \subseteq V$, a set of destination nodes $T \subseteq V$, number of top paths $r$, number of top colors $k$

**Ensure:** A set of $k$-colors such that the minimal reachability between any pair of $s$ and $t$ is maximized.

1: convert $\mathcal{G}$ to a weighted multigraph $\mathcal{G}'$
2: **for** $\langle s, t\rangle$ in $S \times T$ **do**
3: get top-r most reliable paths for $\langle s, t\rangle$ on graph $\mathcal{G}'$
4: **end for**
5: find a minimum color set $C_b$ using Algorithm 4.4
6: get top-$k$ color set $C_{top}$ using Algorithm 4.5
7: **return** $C_{top}$

### 4.3.2 Baselines

We modify the estimate function of the baselines to adapt them to the Max-Minimum problem.

**Baseline 1**

Baseline 1 for Max-Minimum estimates the colors individually by using the minimal estimation over all $\langle s, t\rangle$-pairs. Total time complexity is $O(|C|K(n + e)(|S| \cdot |T|) + k \log |C|)$.

**Algorithm 4.4** Find Minimum Color Set

**Require:** A set $\mathcal{S}_{all}$ of all top-$r$ most-reliable path sets $\mathcal{P}_{s,t}$ between each $\langle s,t\rangle \in S \times T$ , the number of colors $k$

**Ensure:** A minimum set of colors $C_b$ such that each pair $\langle s,t\rangle$ is connected by considering only edges with colors in $C_b$

$C_b = \emptyset$
$\mathcal{S}_{remain} = \mathcal{S}_{all}$
**while** $\mathcal{S}_{remain} \neq \emptyset$ **do**
  $\mathcal{P}_{remain} = \bigcup \mathcal{S}_{remain}$
  $P^* = \arg\min_{P \in \mathcal{P}_{remain}} |C(P) \setminus C_b|$
  $C_b = C_b \cup C(\{P^*\})$
  **while** $|C_b| > k$ **do**
    **for** $c$ in $C_b$ **do**
      $\mathcal{S}_{done} = \mathcal{S}_{all} \setminus \mathcal{S}_{remain}$
      find at least one path $p'$ with $C(p') \subseteq C_b \setminus \{c\}$ for each path set in $\mathcal{S}_{done}$
      remove $c$ from $C_b$ if successful
    **end for**
    **if** not possible to remove enough colors **then**
      **return** any $k$-colors
    **end if**
  **end while**
  $\mathcal{S}_{remain} = \mathcal{S}_{remain} \setminus pathset(P^*)$
**end while**
**return** $C_b$

**Algorithm 4.5** Iterative Path Inclusion for Max-Minimum

**Require:** A set $G_{pairs}$ of uncertain graphs $\mathcal{G}_{\mathcal{P}}$, each induced by the top-$r$ most-reliable path set $\mathcal{P}$ between each $\langle s,t\rangle \in S \times T$ , the number of colors $k$, an initial set of colors $C_b$ that guarantees a basic connectivity in all $\mathcal{G}_{\mathcal{P}}$

**Ensure:** A subset of colors $C_1 \subseteq C$ that maximizes the minimal $Rcol_{\mathcal{G}_{\mathcal{P}}}(s,t,C_1)$ over all $s$-$t$-pairs, while $|C_1| \leq k$

$C_1 = C_b$
**while** $|C_1| < k$ **do**
  $\mathcal{G}^* = \arg\min_{g \in G_{pairs}} Rcol_g(s_g, t_g, C_1)$
  $P^* = \arg\max_{P \in \mathcal{P}(\mathcal{G}^*) \setminus \mathcal{P}_1} Rcol_{\mathcal{G}^*}(s,t,C(\{P\}) \cup C_1)$, such that $|C(\{P\}) \cup C_1| \leq k$
  $C_1 = C_1 \cup C(\{P^*\})$
**end while**
output $C_1$

**Baseline 2**
The modifications in Baseline 2 are similar to those in Baseline 1. Each color in each iteration is estimated using the minimum, with a total complexity of $O(|C|kK(n+e)(|S|\cdot|T|))$.

## 4.4 Max-Connectivity

For the Max-Connectivity maximization problem we do not distinguish between source and destination nodes. There is only a set of equal terminal nodes. The objective of the top-$k$ Max-Connectivity problem is to find colors such that all nodes in the terminal set are most probably connected. The single-source/destination problem can be understood as a special case of the Max-Connectivity problem with a peer set containing two nodes. This problem is fundamentally different from the ones we have seen so far, as we do not handle individual $\langle s,t\rangle$-pairs, and therefore cannot reuse the shortest path algorithms.

**Application**
*Social events:* A possible application could be the organization of a social event e.g. a party. A host organizes a themed party with one or more themes (colors). He wants that the guests (vertices) have common interests (edges) about a theme to promote interaction between his guests. By applying the Max-Connectivity problem, the host could find the top themes for his party.

**Problem formalization**
In Chapter 2 we saw that the connectivity for a single $\langle s,t\rangle$-pair is defined as:

$$R_{C_1}(s,t) = \sum_{G \sqsubseteq \langle \mathcal{G}, C_1\rangle} [Pr(G|C_1) \times I_G(s,t)]$$

We introduce $Rcon_{C_1}(S)$ as the generalized version of $R_{C_1}(s,t)$, which accepts an arbitrary number of terminal nodes. $Rcon_{C_1}(S)$ is defined as follow:

$$Rcon_{C_1}(S) = \sum_{G \sqsubseteq \langle \mathcal{G}, C_1\rangle} [Pr(G|C_1) \times \prod_{\langle s,t\rangle \in S\times S} I_G(s,t)]$$

which is the probability that all nodes in $S$ are connected when only edges with colors in $C_1$ are considered. Using $Rcon_{C_1}(S)$, we formally note the Max-Connectivity maximization problem in a graph $\mathcal{G} = (V, E, C, P)$ as follows:

$$\begin{aligned} &\arg\max_{C_1 \subseteq C} Rcon_{C_1}(S) \\ &\text{such that } |C_1| = k \end{aligned}$$

where $S \subseteq V$ is the set of terminal nodes and $C_1 \in C$ is the set of selected colors that solves Max-Connectivity top-$k$ colors problem.

### 4.4.1 Heuristic

Due to the similarity to the single-source/destination problem, we can reuse the outline of its heuristic solution. Instead of working with shortest paths, we work with minimal Steiner trees (Algorithm 4.6). A Steiner tree is a tree in a weighted graph G with a set of terminal nodes $S \subset V$ that spans all nodes of $S$. A minimal Steiner tree is a Steiner tree where the sum of its edge-weights is minimal. Figure 4.4 shows an example of a minimal Steiner tree that spans the nodes $s_1, s_2, s_3, s_4$.

**Find Top-$r$ Steiner trees**
We begin by converting the graph into a weighted graph with single-colored edges (Algorithm 2.2). Then we find the top-$r$ minimal Steiner trees on the weighted graph. Unfortunately, the *find top-r Steiner trees* problem is NP-hard, so it is not possible to find an exact solution within a reasonable amount of time. For a good approximation we use the DPBF-$r$ algorithm [DING07].

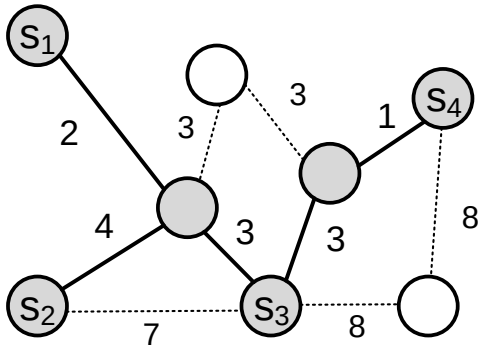



Figure 4.4: Minimal Steiner tree on weighted graph

**Iterative Steiner tree inclusion**
The iterative Steiner tree inclusion algorithm is structurally identical with the *Iterative Path Inclusion* algorithm (2.3). The sole difference is, that we work on a tree set instead of a path set. The function $Rel_{\mathcal{T}_1}(S)$ estimates the connectivity of the terminal nodes $S$ on a graph $\mathcal{T}_1$ induced by the included Steiner trees.

The naive approach to get $Rel_{\mathcal{T}_1}(S)$ is to estimate every single s-t pair ($Rel_{\mathcal{T}_1}(s,t)$):

$$Rel_{\mathcal{T}_1}(S) = \sum_{s \in S} \sum_{t \in S} Rel_{\mathcal{T}_1}(s,t)$$

The naive approach is biased to a group of strongly connected nodes and neglects weekly connected ones and is rather slow. We can improve the performance by fixing one node $s \in S$:

$$Rel_{\mathcal{T}_1}(S) = \sum_{t \in S} Rel_{\mathcal{T}_1}(s,t)$$

A fixed $s$ is computationally fast but the result is depending on the selection of $s$. To avoid a bias we use the minimal pair:

$$Rel_{\mathcal{T}_1}(S) = \min_{s \in S} \sum_{t \in S} Rel_{\mathcal{T}_1}(s, t)$$

*Min* also considers weekly connected nodes and leads to better results regarding to Max-Connectivity.

The Max-Connectivity problem requires a guaranteed connection between all nodes, therefore the *Min* approach is the best choice. To optimize the performance in undirected graphs, we can combine *Min* with a fixed $s$ by fixing $s^*$ and minimize over $t$.

$$Rel_{\mathcal{T}_1}(S, s^*) = \min_{t \in S} Rel_{\mathcal{T}_1}(s^*, t)$$

In a directed graph, the connection between a pair of nodes might not be bidirectional. Use a fixed node as start might therefore lead to false results. Therefore, directed graphs require to vary over both nodes:

$$Rel_{\mathcal{T}_1}(S) = \max_{s \in S} Rel_{\mathcal{T}_1}(S, s^*)$$

*Hardness:* As a generalization of the iterative path inclusion problem, the tree inclusion problem is also NP-hard. It is also neither sub-modular nor super-modular. [KHAN15]

**Time complexity**

The complexity of find the top-$r$ Steiner trees is determined by DPBF-$r$ which is $O(3^l n + 2^l((l + \log n)n + m))$ where $l = |S|$ is the number of terminal nodes in $S$.

We can get $Rel_{\mathcal{T}_1}(S, s^*)$ in a single Monte-Carlo sampling with some modifications, which is $O(r^2(n' \log l + e')K)$ for undirected graphs. In the directed case, we find $Rel_{\mathcal{T}_1}(S)$ in $O(lr^2(n' \log l + e')K)$.

**Difficulties**

On large datasets, DPBF-$r$ is rather slow and uses a lot of memory. In order to limit the execution time, we consider to add an iteration limit as we have for H-MGD.

**Algorithm 4.6** Tree-based heuristic for Max-Connectivity

**Require:** Uncertain graph $\mathcal{G} = (V, E, C, P)$, a set of terminal nodes $S \subseteq V$, number of top paths $r$, number of top colors $k$

**Ensure:** A color set $C_1$ that maximizes the edge-color-constrained reliability $Rcon_{C_1}$

1: Convert the uncertain graph $\mathcal{G}$ into an weighted graph $\mathcal{G}'$ (2.2)
2: Find the top-$r$ minimal Steiner trees $\mathcal{T}_1$ for $S$ on $\mathcal{G}'$ using DPBF-$r$ algorithm.
3: Use the Iterative Tree Inclusion algorithm to obtain the top-$k$ color set $C_1$ from $\mathcal{T}_1$.
4: Add random colors from $C$ to $C_1$ until $|C_1| = k$.
5: **return** $C_1$

### 4.4.2 Baselines

To upgrade the single baselines for the Max-Connectivity problem, we replace the estimation function with $Rel_{\mathcal{T}_1}(S)$.

**Baseline 1**
Time complexity of Baseline 1 is $O(|C|K(n \log l + e))$ for undirected and $O(|C|K(n \log l + e)|S|)$ for directed graphs.

**Baseline 2**
Baseline 2 has a time complexity of $O(|C|kK(n \log l + e))$ for undirected and $O(|C|kK(n \log l + e)|S|)$ for directed graphs.

# 5
# Single-Source/Destination Experiments

In the next two chapters we measure the performance of our proposed algorithms under different parameters. We use four different datasets, each representing a different domain of real world data: Freebase, Biomine, Flixster, DBLP. A detailed description of the datasets can be found in Appendix A.

The implementation of the algorithms is written in C++. To store and manipulate the graph data we use the LEDA[1] class library, which provides a set of high efficient collection- and graph-datatypes. To run experiments we have a cluster of Linux servers, each with a Intel Xeon L5520@2.26GHz CPU and 24GiB of memory.

In our experiments, we are primarily interested in the execution time and the reliability of the algorithms under different parameters. To determine the reliability of the found colors, we reuse the same MC-sampling as we use to find the best colors in the baselines. To keep the balance between performance and accuracy, we set the number of MC-samples to 50 for all experiments.

Before we start experimenting with multi-source/destination queries in chapter 6, we run some tests for single-source/destination, detailed in this chapter. One purpose is to reproduce the results from A. Khan *et al.* [KHAN15], another to learn about the characteristics of the algorithms for each dataset. This is relevant because the pairwise maximization problems heavily depend on single pair queries.

The heuristic and the baseline algorithms use MC-sampling to estimate the reliability, a higher sampling size $K$ means a better reliability but increases the execution time. We use $K$=1000 for single/source experiments as A. Khan *et al.* did in their paper. To keep the experiment time down, we reduce $K$ to 100 when we know that algorithms are slow under the given parameters. To keep the results comparable we extrapolate their execution time be equivalent to $K$=1000 for all experiments. According to our tests, reducing $K$ means a loss in reliability between 0 and 0.03 which does not change the outcome significantly.

[1] http://www.algorithmic-solutions.com/leda/

## 5.1 Varying Datasets

In the first experiment we measure the reliability and the execution time of single-source/destination queries on different datasets.

### 5.1.1 Experiment Setup

We compare the two baselines with the two variations of the heuristic algorithm. We run the experiments on all four datasets: Freebase, Biomine, Flixster, DBLP. The remaining parameters are constant. As we aim to reproduce the results of A. Khan *et al.*, we reuse the values for $k$ (5), $r$ (20) and $K$ (1000) from that paper. To keep the time down we reduce the number of sampling loops $K$ for Baseline 2 to 100 if necessary.

To find an accurate average for the run time and the reliability of each algorithm we run a set of 500 queries (352 for Flixster, 200 for DBLP) on each dataset. The queries we use are generated by selecting a pair of two random nodes from the graph. The source and the destination are guaranteed to be connected. See appendix A.2 for more details.

**Experiment parameters**:

| | |
|---|---|
| Datasets: | Freebase, Biomine, Flixster, DBLP |
| Number of colors: | $k$ = 5 |
| Number of shortest paths: | $r$ = 20 |
| MC-samples in Baseline 1: | $K$ = 1000 |
| MC-samples in Baseline 2: | $K$ = 100, Freebase: 1000 |
| Source/destination nodes: | 1:1 |
| Number of queries: | 500, Flixster: 352, DBLP: 200 |

### 5.1.2 Results

Table 5.1 shows the outcome of the experiment. We observe that the datasets highly differ in how the algorithms perform in execution time and reliability.
*Freebase:* The heuristic algorithms beat the two baselines in terms of reliability. Baseline 2 is significantly slower than Baseline one. In terms of run time, both heuristics beat the Baseline 1 by two orders of magnitude
*Biomine:* The reliability of H-MGD is on the same level as Baseline 1 while H-LE is significantly better than these two while only slightly worse than Baseline 2. We noted that H-MGD hits its execution limit to quick, which decreases its reliability, therefore the limit had to be increased to 5 million loops to be able to beat Baseline 1.
*Flixster:* Flixster is the sole dataset where H-MGD has a better reliability than H-LE. The heuristics have about the same reliability as Baseline 1. Surprisingly, Baseline 1 is really fast compared to the other datasets, it even outperforms H-MGD. Only H-LE can beat Baseline 1 in terms of reliability and run time.
*DBLP:* Baseline 1 and H-LE have similar reliability, only Baseline 2 can beat them. Similar to Biomine, H-MGD will abort before it can find enough shortest paths. Therefore the reliability falls behind H-LE and Baseline1.

*Summary:* We can reproduce the results of A. Khan *et al.* quite well except for Baseline 1 on Flixster. The heuristics in general are some orders of magnitude faster that the baselines,

while Baseline 2 usually has the best reliability followed by H-LE.

| | Reliability | | | | Time (sec) | | | |
|---|---|---|---|---|---|---|---|---|
| **DataSet** | **BL1** | **BL2** | **H-MGD** | **H-LE** | **BL1** | **BL2** | **H-MGD** | **H-LE** |
| Freebase | 0.15 | 0.15 | 0.17 | 0.17 | 1.38 | 43 | 0.01 | 0.02 |
| Biomine | 0.16 | 0.37 | 0.18 | 0.31 | 1220 | 26217 | 34.68 | 5.27 |
| Flixster | 0.46 | 0.62 | 0.50 | 0.48 | 1.43 | 809 | 1.65 | 0.06 |
| DBLP | 0.06 | 0.10 | 0.03 | 0.07 | 420 | 8301 | 2.91 | 4.76 |

Table 5.1: Single-source/destination results

### 5.1.3 Analysis

**Shortest paths**
With H-MGD and H-LE we have two different approaches to find the shortest paths. We focus on H-LE as it is in general superior to H-MGD.

H-LE finds at least one shortest path for any query, this is not surprising as we selected the queries to have a connection between the source and the destination. In most cases H-LE will return 20 paths, with a few exceptions when only a single path could be found. The average number of paths found are 19.2 for Freebase, 19.2 for Biomine, 19.9 for Flixster and 19.1 for DBLP.
The usefulness of a path is determined by its reliability and colors. We realize that some properties of the path lowers it usefulness for color selection:

- Too many colors in path: If a single path has already more than $k$ colors, it cannot be used during the Iterative Path Inclusion. From our data we observe that when the (top-1) shortest path exceeds the color limit, the other paths do too in almost all cases. This could be an indication that the source and the destination are too far away from each other to find meaningful $k$ colors.
- Path contains cycles (Figure 5.2): Eppstein allows paths to have cycles. This can become a problem when an $(i+1)$th-path just adds an additional cycle to an $i$th-path, as we could not find new colors from this kind of paths. Cycles with moderate length will eventually become too costly and alternative paths will be considered as we increase $r$. However, if the cycle cost is very low, new paths will keep adding cycles and increasing $r$ will not help.

**Freebase**
Freebase is by far the largest graph with over 28M nodes and 48M edges, thus all algorithms having a low run time is rather surprising. However, the average node degree is very low and, on average, there are only a few routes from the source to the destination. Hence their reachability is most often determined only by the shortest path. In consequence, the MC-sampling becomes very effective. The lack of alternative routes causes the reliability to be rather small, even when the average probability of the edges is the highest among our four datasets. Freebase contains about 5000 different colors, worsening the *cold start* problem of Baseline 2, which eventually does not perform better than Baseline 1.

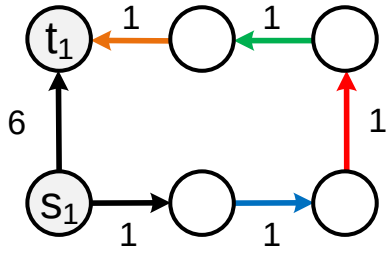



Figure 5.1: Hops in shortest path

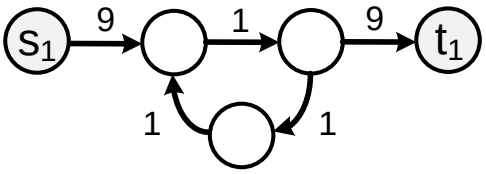



Figure 5.2: Shortest paths with low cost cycle

The heuristics are performing way better than the baselines in terms of colors found. Nonetheless, 38% of the queries evaluate to zero as the reliability of the shortest path is below the evaluation precision. The paths found by H-LE are not very diverse and many of them contain cycles. We will address this issue later in Section 5.4.

**Flixster**
The Flixster graph is very different from Freebase. Its nodes are highly interconnected and there are only 10 different edge colors. With a relatively high $k$ of 5, there is a rather high *random base reliability*, this means that even randomly selected colors have a high reliability. From the results we see that Baseline 2 is much more reliable than the others. We see this trend also when we look at the individual queries: In 348 out of 352 queries, Baseline 2 finds at least one color. H-MGD does for 235, Baseline 1 for 145 and H-LE for only 142 queries. We inspect the individual shortest paths retrieved by H-LE to find two reasons for its bad performance.

- The top-$r$ shortest paths have more than $k$-colors: Surprisingly this is also true for a pair of nodes that is actually close to each other. The nodes of Flixster are highly interconnected with each other, and many edges have high probabilities that are one or close to one. As a consequence, a shortest path between two nodes can become very long in terms of hops as illustrated in Figure 5.1.
- Paths contain cycles: As in the other graphs, shortest paths at a higher top-$r$-position (Figure 5.2), tend to have cycles. In Flixster these cycles have a very low cost, sometimes even zero, making it impossible to find other, more distinct paths.

*H-LE vs. H-MGD*: Flixster is the only graph where H-MGD outperforms H-LE. This fact is directly related to the issues with H-LE. H-MGD is designed to accept *false* shortest paths (Section 3.1.3). These *"shortest paths"* have in general less hops then the true ones. Eventually these paths do not exceed the color limit that often, which allows H-MGD to find more colors. Also, MGD's paths tend to have less cycles. In conclusion, short paths with fewer hops are sometimes better than paths with lower cost.

**Biomine**
For Biomine, we focus on the comparison between H-LE and Baseline 2. Baseline 2 has a better overall reliability but finds colors in only 74% of the queries, while H-LE does in 91%.

In conclusion, the overall quality of the selected colors must be better in Baseline 2. We do a *case study* on some sample queries where H-LE performs especially bad. Table 5.2 shows the colors in the order they have been found and the estimated reliability. In each query, 2 to 3 colors are shared between H-LE and Baseline 2. Interestingly the first two colors selected by Baseline 2 are rarely selected by H-LE. In most cases these colors are not even covered by a single path in the top-$r$ set. The colors of H-LE are mostly determined by the shortest path, as it is the most likely to be included first during the *Iterative Path Inclusion*.

The Biomine graph has many highly interconnected nodes. Between any $s$ and $t$ there is a complex network of possible routes. A lot of these individual routes have low probability, but combined, they provide a better reachability as one of the top-$r$ paths alone. In consequence, picking only a few top paths is not as effective as in other datasets like Freebase.

| | **Baseline 2** | | **H-LE** | |
|---|---|---|---|---|
| **Query** | **Colors** | **Reliability** | **Colors** | **Reliability** |
| 1 | 5 12 14 8 17 | 0.92 | 4 14 8 9 17 | 0.68 |
| 2 | 9 14 17 4 | 0.98 | 14 13 4 18 | 0.58 |
| 3 | 6 8 12 14 11 | 0.68 | 4 13 18 14 12 | 0.28 |
| 4 | 4 13 18 14 12 | 0.82 | 9 0 1 14 8 | 0.42 |
| 5 | 6 17 8 9 3 | 0.68 | 9 14 18 4 | 0.16 |

Table 5.2: Case study Biomine

**DBLP**

The DBLP graph has low probability edges and many different colors. The colors affect Baseline 2's cold start problem. Therefore Baseline 2, along with Baseline 1, doesn't find any colors for 73% of the queries, but Baseline 2 benefits from including colors from parallel edges along the path from the source to the destination, resulting in a higher average reliability.
In 80% of the DBLP-queries, all top-20 shortest paths have more than 5 colors, leaving H-LE without finding any colors. For the remaining 20%, the heuristic colors have a reliability of 0.36.

**Lazier Eppstein efficiency**

In Section 3.1.2, we describe the modified Eppstein algorithm that would reduce the number of nodes for which we need to calculate the shortest path. The modification itself cannot guarantee any theoretical time reduction, so we rely on actual measurements to evaluate the improvement. For Freebase, H-LE had to calculate the shortest paths to 101 nodes on average, which is 0.0004% of all nodes (28'483'132). The median was 42 and maximum 437. For Biomine, the average is 836'578 (80% of 1'045'414) with a median of 882'578 and a maximum of 981'418. DBLP has an average of 529'389 (42% of 1'251'335), a median of 572'976 and the maximum of 1'113'239. The Flixster graph is directed, hence a reduction is not possible.

We state that the reduction ratio is between 20% and 99%. The ratio for Freebase is a positive surprise, as without the reduction, H-LE would not be able to beat Baseline 1 in execution time.

**Cycled paths issue**
Paths with cycles reduce the effectiveness of the heuristic algorithm. To address this problem we tried to remove 2-hop cycles ($A \longrightarrow B \longrightarrow A$) in H-LE. The first attempt was to completely ignore these routes. We tested the modification on Freebase with the result that the cycled path were removed but no replacements were found, leaving the average number of returned shortest path close to one. The overall reliability remained the same as without the modification.

The second attempt was to not include those paths in the top-$r$ set, but to further explore side paths. That lead to a highly increased run time which could not compete with Baseline 1.
In conclusion, there is no (easy) way to even remove 2-hop cycles from H-LE.

## 5.2 Varying top-k

In the second experiment we study the effect of varying the number of colors on Freebase and Flixster.

### 5.2.1 Experiment Setup

We vary the number of colors $k$ from 5 to 20. Freebase has over 5000 distinct edge colors, so increasing $k$ is expected to have a very low impact on the results. Flixster however has only 10 different colors, the impact there is expected to be significant. We use the same queries and parameters as in the previous experiment.

**Experiment parameters**:

| | |
|---|---|
| Datasets: | Freebase, Flixster |
| Number of colors: | Freebase: $k$ = 5, 10, 15, 20 |
| | Flixster: $k$ = 5, 6, 7, 8, 9, 10 |
| Number of shortest paths: | $r$ = 20 |
| MC-samples in Baseline 1: | $K$ = 1000 |
| MC-samples in Baseline 2: | $K$ = 100 |
| Source/destination nodes: | 1:1 |
| Number of queries: | Freebase: 500, Flixster: 352 |

### 5.2.2 Results

In Table 5.3 we see the outcome for the different values of $k$ for Freebase. As expected, the data shows that there is no significant difference in the reliability on any of the tested algorithms. The same holds true for the execution time with the exception of Baseline 2 where $k$ has by design a much larger influence on the time complexity.

Table 5.4 shows the results for Flixster. The reliability gap between Baseline 2 and the other becomes smaller with increasing $k$. Flixster has a total of 10 colors. Setting $k$=10 is equivalent of having no color restriction, which is intuitively the same for all algorithms.

| | Reliability | | | | Time (sec) | | | |
|---|---|---|---|---|---|---|---|---|
| **Top-k** | **BL1** | **BL2** | **H-MGD** | **H-LE** | **BL1** | **BL2** | **H-MGD** | **H-LE** |
| 5 | 0.151 | 0.153 | 0.170 | 0.172 | 1.38 | 43.73 | 0.01 | 0.02 |
| 10 | 0.152 | 0.154 | 0.166 | 0.167 | 1.50 | 112.44 | 0.01 | 0.03 |
| 15 | 0.154 | 0.154 | 0.169 | 0.166 | 1.50 | 179.65 | 0.01 | 0.03 |
| 20 | 0.150 | 0.154 | 0.170 | 0.171 | 1.50 | 248.51 | 0.01 | 0.03 |

Table 5.3: Varying top-k on Freebase

| | Reliability | | | | Time (sec) | | | |
|---|---|---|---|---|---|---|---|---|
| **Top-k** | **BL1** | **BL2** | **H-MGD** | **H-LE** | **BL1** | **BL2** | **H-MGD** | **H-LE** |
| 5 | 0.46 | 0.62 | 0.50 | 0.48 | 1.43 | 809 | 1.65 | 0.06 |
| 6 | 0.53 | 0.67 | 0.59 | 0.59 | 1.62 | 1462 | 2.74 | 0.20 |
| 7 | 0.61 | 0.70 | 0.66 | 0.65 | 1.68 | 1747 | 2.25 | 0.22 |
| 8 | 0.67 | 0.71 | 0.69 | 0.70 | 1.72 | 1896 | 3.88 | 0.28 |
| 9 | 0.71 | 0.73 | 0.72 | 0.73 | 1.76 | 2133 | 2.72 | 0.28 |
| 10 | 0.75 | 0.74 | 0.74 | 0.74 | 1.74 | 2089 | 2.79 | 0.25 |

Table 5.4: Varying top-k on Flixster

### 5.2.3 Analysis

For the 500 individual queries in Freebase, the average number of colors found by H-LE is 1.8 and the maximum 3. H-MGD has the same maximum and a slightly less average of 1.7. Baseline 1 finds on average 0.5 non-random colors. For Baseline 2, it is difficult to exactly determine how many of the selected color have significant influence on the reliability. Our analysis shows that its average is around 1.5. Hence, increasing $k$ above 3 will have no effect on the reliability. The Freebase graph contains over 5000 different colors, so have more random colors in the color set will not increase to reliability either as chances are low to accidentally pick a "good" color.

For Flixster, with a higher $k$ the heuristics find significantly more colors. But increasing $k$ leads also to a higher *random base reliability*. Even when Baseline 1 does not find more colors on higher $k$, the average reliability increases about the same as for the other algorithms.

## 5.3 Varying hops

Previously, we used randomly selected source/destination pairs where the distance between both nodes may highly vary. For this experiment, we fix the number of hops to certain distance.

### 5.3.1 Experiment Setup

We fix the number of hops between the source/destination to a value of 2, then reproduce the experiment twice, with distances of 4 and 6.

The queries are generated by selecting a pair of two random nodes where the distance between the two nodes is exactly $n$-hops. See appendix A.2 for more details.

**Experiment parameters**:

| | |
|---|---|
| Datasets: | Biomine |
| Number of colors: | $k$ = 5 |
| Number of shortest paths: | $r$ = 20 |
| MC-samples in Baseline 1: | $K$ = 100 |
| MC-samples in Baseline 2: | $K$ = 100 |
| Source/destination nodes: | 1:1 |
| Number of queries: | 500 |

### 5.3.2 Results

Table 5.5 shows the results for different $n$-hops on Biomine. The reliability decreases with an increasing distance. The reliability of Baseline 1 drops above 2 hops when there are fewer single-colored s-t-connections left. H-LE has a better reliability than Baseline 2. Even H-MDG can beat Baseline 2 on 2-hops, but above the execution limit causes a sharp drop. The iteration limit for H-MDG is 1 million. Therefore, unlike in the first experiment, H-MGD is faster than H-LE.

| **Distance** | **Reliability** | | | | **Time (sec)** | | | |
|---|---|---|---|---|---|---|---|---|
| **hops** | **BL1** | **BL2** | **H-MGD** | **H-LE** | **BL1** | **BL2** | **H-MGD** | **H-LE** |
| 2 | 0.45 | 0.75 | 0.80 | 0.83 | 346 | 9798 | 1.35 | 4.90 |
| 4 | 0.08 | 0.38 | 0.20 | 0.64 | 548 | 23140 | 2.79 | 5.58 |
| 6 | 0.02 | 0.17 | 0.02 | 0.30 | 406 | 29135 | 3.64 | 5.37 |

Table 5.5: Varying hops on Biomine

### 5.3.3 Analysis

**Baselines**
Unsurprisingly, the reliability for Baseline 1 falls quickly with an increasing number of hops as there are fewer single-colored paths between the source and the destination: 260 queries with at least one color found for 2-hops, 21 for 4-hops and 9 for 6-hops. We observe the same for Baseline 2 due to the *cold start* effect but with a smaller influence. Queries with colors found by Baseline 2: 381 for 2-hops, 270 for 4-hops and 175 for 6-hops (Figure 5.3).

**Heuristic**
Both heuristic algorithms start with a good reliability on the same level as Baseline 2. H-LE find as many colors in 2-hop (500) as in 4-hop (469). Even for 6-hop the yield is good (355). H-MDG is also good for lower hops but becomes worse quickly with an increasing distance as the execution limit prohibits to find enough shortest paths. By increasing the limit we could bring the reliability up but at some point, it would become much slower than H-LE.

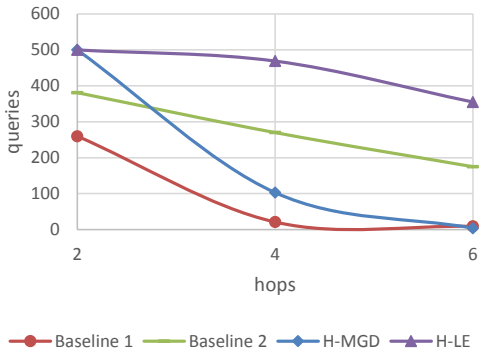



Figure 5.3: $n$-hop queries

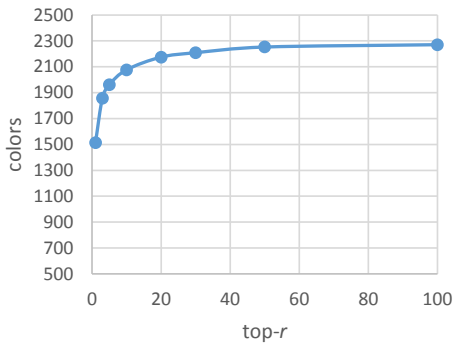



Figure 5.4: top-$r$ total colors

## 5.4 Varying top-r

In the last experiment on single-source/destination queries, we examine the influence of the number $r$ of shortest paths on the heuristic algorithms. Increasing $r$ may potentially improve the reliability as the *Iterative Path Inclusion* has more paths to choose from.

### 5.4.1 Experiment Setup

We vary the number of paths obtained by the shortest path algorithm between 1 and 100. $r$=1 means we only obtain the shortest path. We list the results for H-LE as it has the better overall performance.

**Experiment parameters**:

| | |
|---|---|
| Datasets: | Biomine, Freebase, Flixster |
| Number of colors: | $k$ = 5 |
| Number of shortest paths: | $r$ = 1 to 100 |
| Source/destination nodes: | 1:1 |
| Number of queries: | 500, Flixster: 352 |

### 5.4.2 Results

As shown in Table 5.6, the effect of increasing $r$ diminishes quickly after just a few increments. In the case of Biomine, we have about the same reliability when we use 10 instead of 20 paths or more.

### 5.4.3 Analysis

Figure 5.4 shows the sum of colors found over all 500 queries. We see that the number of colors correlates with the average reliability. After more than 20 paths, there is not much growth in additional colors. For a further explanation, we look at the shortest paths retrieved by H-LE. From the first experiment we know that some paths may contain cycles. These are not useful as they contain the same colors as the cycle free paths. When we filter for those without cycles, we see that with $r$=15 , on average only 50% of the paths are cycle

| Top-r | Reliability | Time |
|---|---|---|
| 1 | 0.27 | 4.29 |
| 2 | 0.29 | 4.26 |
| 3 | 0.31 | 4.30 |
| 4 | 0.31 | 4.30 |
| 5 | 0.31 | 4.31 |
| 10 | 0.32 | 4.31 |
| 15 | 0.32 | 4.37 |
| 20 | 0.32 | 5.26 |
| 30 | 0.33 | 5.29 |
| 50 | 0.33 | 5.38 |
| 100 | 0.33 | 5.70 |

Table 5.6: Varying top-r for H-LE on Biomine

free. This ratio get worse as we increase $r$. Table 5.7 illustrates how many of the top paths at each position are cycle free. The case of Biomine, even with a high $r$ we get a decent amount of cycle free paths. However, being cycle free does not mean a paths has more colors. Conclusion: The effect for increasing $r$ gets lower for a larger $r$ as more of the additional paths have cycles.

| Top-r | 1 | 2 | 3 | 4 | 5 | 10 | 15 |
|---|---|---|---|---|---|---|---|
| **Biomine** | 100% | 76% | 67% | 57% | 51% | 41% | 34% |
| **Freebase** | 100% | 12% | 4% | 2% | 2% | 1% | 0% |
| **Flixster** | 100% | 50% | 22% | 15% | 13% | 7% | 3% |

Table 5.7: Cycle free path distribution

**Case study**
For some queries, increasing $r$ does indeed result in more colors. Table 5.8 shows the number of colors in relation to $r$ for an individual Biomine query. At certain points, new paths will be cycled versions of previous paths, but the cycles increase the cost of the path. Eventually a cycle-free path is shorter and new colors are discovered.

| Top-r | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Colors** | 2 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 5 |
| **Cycles** | No | Yes | No | Yes | Yes | Yes | No | No | Yes | No |

Table 5.8: Top-$r$ case study

**Freebase & Flixster**
We omitted the results for Freebase and Flixster in Table 5.6 because it did not show any improvements in the reliability when we increase $r$. In Freebase, only 12% of the second shortest paths are cycle free (Table 5.7) which makes it effectively useless to increase $r$ further beyond 1. The same is true for Flixster beyond 3 paths.

# 6 Multi-Source/Destination Experiments

In this chapter we present the experiments on our four multi-source/destination problems. As we have more parameters to vary, we perform much more measurements. To keep the experiment duration down, we reduce the number of queries for each measurement to 100.

## 6.1 Max-Connectivity on varying datasets

In the first experiment on a multi-source/destination problem, we compare the execution time and reliability of our heuristic solution for Max-Connectivity against the two baselines on different datasets.

### 6.1.1 Experiment Setup

We vary over the four datasets while keeping the other parameters fixed. Finding the top Steiner tree on large datasets can take a very long time. Therefore, we define a maximum limit of iterations to find the trees. We set this limit to 500'000, as a compromise between reliability and execution time that can be used on all datasets.

**Experiment parameters**:

| | |
|---|---|
| Datasets: | Freebase, Biomine, Flixster, DBLP |
| Number of colors: | $k$ = 5 |
| Number of minimal Steiner trees: | $r$ = 20 |
| MC-samples in Baseline 1: | $K$ = 100, Flixster: 1000 |
| MC-samples in Baseline 2: | $K$ = 100 |
| Terminal nodes: | 4 |
| Number of queries: | 100 |

We use a query set that has 4 terminal nodes in each query. The nodes were randomly selected within a 1-hop radius around a central node and a maximal distance between any two terminal nodes of 2.
During the experiment it became clear that the average execution time for Baseline 1 and Baseline 2 on Freebase is too high to get meaningful results in time, but the execution times are distributed uneven. About 75% of the randomly generated queries finish in less than one hour. For the experiment we use only the queries that finish within an hour.

### 6.1.2 Results

Table 6.1 shows the results for Max-Connectivity. The execution times of Baseline 1 and Baseline 2 for Freebase are an estimation of the value if we would include the remaining 25% of queries.

The reliability of the heuristic approach beats Baseline 1 but cannot compete with Baseline 2. The exception is DBLP where also Baseline 1 is better. There is a similar problem as we encountered with H-MGD where an artificial limit stops the algorithm from finding more Steiner trees. We will investigate this issue in the analysis. To prove that the heuristic could do better, we repeated the experiment on a connected sub graph of DBLP (denoted as DBLP-s) which contains about 15% of total the edges and nodes. Without hitting the limit, heuristic can outperform Baseline 1 on DBLP-s.

| | **Reliability** | | | **Time (sec)** | | |
|---|---|---|---|---|---|---|
| **DataSet** | **BL1** | **BL2** | **Heuristic** | **BL1** | **BL2** | **Heuristic** |
| Freebase | 0.01 | 0.10 | 0.04 | *190844* | *1317564* | 640 |
| Biomine | 0.29 | 0.47 | 0.34 | 893 | 37992 | 93 |
| Flixster | 0.48 | 0.67 | 0.54 | 11 | 9629 | 8 |
| DBLP | 0.31 | 0.46 | 0.19 | 1283 | 304598 | 94 |
| DBLP-s | 0.05 | 0.41 | 0.15 | 73 | 4854 | 3 |

Table 6.1: Max-Connectivity results

### 6.1.3 Analysis

**Increasing the limit**
The limit of 500'000 iterations was set before we started the experiment, with the objective to stay below the execution time of Baseline 1. For Freebase, Biomine and DBLP the heuristic is much faster than Baseline 1, thus we can increase the limit on these datasets for a better reliability. We see a constant improvement in the reliability with higher limits (Table 6.2[1]). With a limit of 9.0M, heuristic can beat Baseline 2 on Biomine, and catch up to Baseline 1 on DBLP while still being faster. On Freebase heuristic also catches up to Baseline 2 in terms of reliability, however in more than 75% of the queries, heuristic would be slower than Baseline 2.

[1]The reliability for Freebase 9.0M is extrapolated from a result set of 70 queries

| | Reliability | | | Time (sec) | | |
|---|---|---|---|---|---|---|
| **Limit** | **Freebase** | **Biomine** | **DBLP** | **Freebase** | **Biomine** | **DBLP** |
| 500k | 0.04 | 0.33 | 0.19 | 640 | 93 | 94 |
| 1.5M | 0.05 | 0.38 | 0.27 | 1697 | 173 | 198 |
| 4.5M | 0.06 | 0.47 | 0.29 | 5234 | 252 | 322 |
| 9.0M | *0.10* | 0.71 | 0.33 | 8019 | 310 | 410 |

Table 6.2: Varying heuristic limit

**Steiner Tree reliability**
Depending on the dataset, we are able to find Steiner trees for 4% (Freebase) to 100% (Flixster) of the queries. If at least one Steiner tree is found, the average reliability is 0.97 for Freebase, 0.81 for Biomine and 0.43 for DBLP. In general the reliability when a tree was found is good, however there are some issues:

1. Only the minimal tree is the optimal solution, other trees are only approximations.

2. As in the single-source/destination experiments, we state that many minimal Steiner trees in Flixster have more than 5 colors, hence the heuristic algorithm can't select any colors.

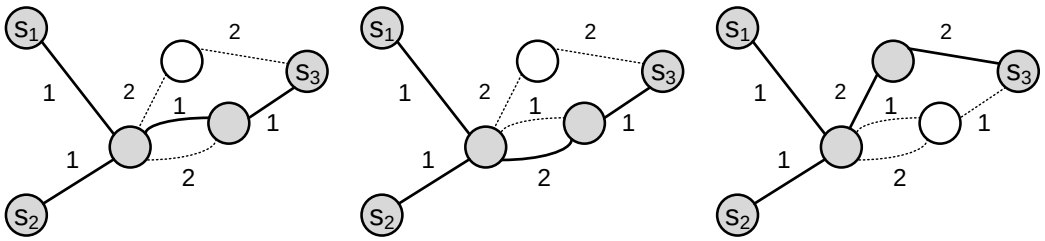
3. The algorithm to find the top-$r$ Steiner trees does not work on multigraphs like Flixster or DBLP. When two nodes are neighbors over multiple edges, only the shortest edge will be considered, missing an opportunity to find relevant colors. This is also true when the nodes are connected over intermediate nodes. This restriction does not apply when one of the nodes is a terminal node. Figure 6.1 illustrates a simple graph and its top-3 minimal Steiner trees. In this example, only (a) and (c) can be found, (b) will be dropped in favor of (a) during the search for minimal trees.

When we only consider queries where Steiner trees are found, Freebase and Biomine also beat Baseline 2 in terms of reliability. However this is not true for Flixster and DBLP due to the issues 2 and 3. Conclusion: Heuristic provides the best balance between reliability and execution time. If only reliability is important, Baseline 2 is the better choice for some datasets. Due to the complexity of the $k$ minimal Steiner tree problem, the heuristic solution is not as scalable as for the single source problem.

**Case study**
We complete our analysis by looking at single Biomine query in detail. The heuristic algorithm found all top-20 Steiner trees and could identify the top-5 colors in 45.9 seconds. The found color set has a reliability of 0.56, the color sets of Baseline 1 and 2 evaluated both to zero. Table 6.3 shows a summary of the Steiner trees. Each tree has at least 3 colors, which caused Baseline 1 and Baseline 2 to fail. 3 of 5 colors are determined by the minimal Steiner tree. The trees position 3 and 4 contain colors which eventually were not selected. If we set $r$=4, the color set would be different. The last color in the top-$k$ set does not appear until the 7th tree. We conclude that unlike the shortest paths, the trees in the top-$r$ set have much more variations. As a consequence, even trees on a high position can influence final color set.

| **Top-tree** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Colors** | 3 | 3 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 4 |
| **Edges** | 3 | 4 | 4 | 4 | 5 | 6 | 7 | 9 | 9 | 5 |
| **Colors in top-$k$** | 3 | 3 | 3 | 3 | 4 | 3 | 5 | 5 | 5 | 3 |
| **Colors not in top-$k$** | | | 1 | 1 | | 1 | | | | 1 |
| **Top-colors discovered** | 3 | | | | 4 | | 5 | | | |

Table 6.3: Case study Max-Connectivity




(a) Top 1 MST (b) Top 2 MST (c) Top 3 MST

Figure 6.1: Example for minimal Steiner trees

## 6.2 AVG, MAX, MIN on varying datasets

In this experiment, we compare the pairwise multi-source/destination algorithms on different datasets with different numbers of source/destination nodes.

### 6.2.1 Experiment Setup

We test the aggregations AVG, MAX and MIN on Freebase, Biomine and Flixster. The test them on queries with a source/destination node number of 2:2, 3:3 and 5:5.

**Experiment parameters**:

| | |
|---|---|
| Datasets: | Freebase, Biomine, Flixster |
| Number of colors: | $k$ = 5 |
| Number of shortest paths: | $r$ = 20 |
| MC-samples in Baseline 1: | $K$ = 100 |
| MC-samples in Baseline 2: | $K$ = 100 |
| Source/destination nodes: | 2:2, 3:3, 5:5 |
| Number of queries: | 100 |

For Freebase, due to the long execution time for Baseline 1 and Baseline 2, we filtered queries which took more than 2 hours to complete, like we did in the previous experiment, otherwise we wouldn't be able to present meaningful results. Thus the Freebase samples consists of only 30 to 50 queries. The execution times in the results are an extrapolation that includes the estimated execution time of queries we sorted out.

### 6.2.2 Results

Tables 6.4[2], 6.5[3], 6.6 show the results for the different aggregations. The ranking of the algorithms in terms of reliability over the various datasets is the same as in the single-source/destination experiments: Baseline 2 is best for Biomine, Flixster and the heuristic for Freebase. Because the maximum distance between the nodes is limited to 2, we see higher reliabilities for Freebase and Biomine. Flixster however is not affected. Like in the single-source experiments the heuristics complete faster than the baselines except for Flixster.

When we compare the execution time of H-LE on Freebase to the single-source/destination experiment (Table 5.1), we would expect a value in the same magnitude as H-MGD, which is not the case. Due to a different distribution of the terminal nodes, the Lazier Eppstein algorithm has now to build the shortest path for significantly more nodes. In the previous experiments the average was 100 (Section 5.1.3), here the average is 24M, 85% of all nodes.

Because of the small distance between the pair nodes, the reliability of H-MGD is about the same as H-LE and, due to the iteration limit, the execution time depends less on the graph size. For large datasets H-MGD could be a viable alternative to H-LE.

### 6.2.3 Results AVG

**Results**
We see that the number of source/destination nodes does not have a big influence on the AVG reliability. The only exception is Freebase 2:2. As this query set was filtered for execution time, the remaining queries could have a bias, which would explain this outliner. We also note that the execution time for the heuristics increases disproportionately with the number of sources.

**Analysis**
The reliability of Max-Average correlates directly with the reliability of the individual source/destination-pairs. Therefore the algorithms have the same ranking order in reliability as they do in the single-source case. In H-LE and H-MGD, the *Iterative Path Inclusion* has to iterate over $r_{ipi} = r|S \times T|$-paths which leads to a disproportional execution time for queries with many nodes as the time complexity is proportional to $r^2_{ipi}$.

We see that the heuristic reliability for queries with many nodes (like 5:5) becomes slightly worse in relation to the baselines. This is because in the heuristic algorithm, we select paths, not colors. Most paths have 2 to 3 colors. With $k$=5 this means, after we select the first path, we have only 2 colors left, which limits the selection of the remaining paths and leads to a bias for strongly connected pairs and eventually reduces the reliability of the selected colors.

[2]Freebase has varying sample sizes: (AVG/MAX/MIN) 2:2=30/38/59, 3:3=29/39/61, 5:5=28/37/75
[3]Biomine has varying sample sizes: 3:3-AVG=73, 3:3-MAX=84, 5:5-AVG=29, 5:5-MAX=53

### 6.2.4 Results MAX

**Results**
The MAX reliability increases as we have more source/destination nodes. The gap in the reliability between the algorithms is not as big as in AVG or MIN and gets smaller with more nodes. With 5 source and 5 destination nodes, there is virtually no difference between the algorithms. The execution time is nearly proportional to the number of node-pairs.

**Analysis**
As we would expect, the maximum increases as we can choose from a larger set of node-pairs. We do a cross check with the results from the first single-source experiment. Assuming the reliabilities of the node-pairs are independent and uniformly distributed, MAX 5:5 should roughly be equivalent to the 0.96 percentile of the previous experiment (Freebase=0.74, Flixster=1, Biomine=0.98 for all algorithms). We see these values match up pretty well with the results from this experiment.

### 6.2.5 Results MIN

**Results**
The results for MIN maximization differ highly for all datasets. Except for Flixster, the heuristic solutions have a better reliability than the baselines and the reliability gap becomes smaller with more source/destination nodes. The execution time for Baseline 1 and Baseline 2 is lower than for MAX and AVG for the same queries, because we can skip estimating the remaining node-pairs when we get one with a zero reliability. H-LE and H-MGD benefit less of this effect as they spend most of the time to find the shortest paths.

**Analysis**
*Freebase:* Baseline 1 and Baseline 2 can find colors only for a few queries where all node-pairs have a single-colored path with the same color. However, these queries evaluate with a low average reliability for Baseline 1, whereas Baseline 2 finds additional colors that improve the result. The paths found by the heuristics have, on average, only a few colors (2-3) and these are very similar for each pair. This allows the MIN heuristic to find a common color set that evaluates well. With more pairs, such common colors are less likely to be present.
*Flixster:* Baseline 1 finds almost no colors for MIN for any number of source nodes, so the result is solely determined by the *random base reliability*. The heuristics do find colors for 2:2, however their evaluation is not different from a set of randomly selected colors. For 3:3 and 5:5, the heuristics can't find any colors. Baseline 2 profits from the low number of total colors which effectively increases the probability to overcome the cold start problem by chance. With a larger color set like in Freebase, it would not perform better than Baseline 1.
*Biomine:* For 2:2, the heuristics find colors for almost all queries. Like Freebase, the shortest paths for each pair often contain 1 to 3 colors with a low color variation among the pairs. There are also many single-colored paths; Baseline 1 finds at least one color for 43% (2:2) to 31% (5:5) of the queries. Baseline 2 benefits from the low number of total colors to overcome the cold start problem.

| | | Reliability | | | | Time (sec) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **BL1** | **BL2** | **H-MGD** | **H-LE** | **BL1** | **BL2** | **H-MGD** | **H-LE** |
| 2:2 | AVG | 0.46 | 0.51 | 0.59 | 0.59 | *26000* | *740000* | 14 | 311 |
| | MAX | 0.61 | 0.64 | 0.71 | 0.72 | *24000* | *656000* | 15 | 191 |
| | MIN | 0.01 | 0.08 | 0.28 | 0.32 | *2500* | *62000* | 11 | 203 |
| 3:3 | AVG | 0.34 | 0.37 | 0.39 | 0.36 | *65000* | *1495000* | 52 | 325 |
| | MAX | 0.69 | 0.72 | 0.73 | 0.71 | *47000* | *1286000* | 30 | 255 |
| | MIN | 0.01 | 0.02 | 0.10 | 0.09 | *5300* | *117000* | 22 | 261 |
| 5:5 | AVG | 0.34 | 0.36 | 0.36 | 0.38 | *99000* | *3028000* | 110 | 612 |
| | MAX | 0.78 | 0.78 | 0.80 | 0.79 | *105000* | *2651000* | 118 | 551 |
| | MIN | 0.01 | 0.02 | 0.04 | 0.05 | *13000* | *155000* | 62 | 581 |

Table 6.4: Results for Freebase

| | | Reliability | | | | Time (sec) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **BL1** | **BL2** | **H-MGD** | **H-LE** | **BL1** | **BL2** | **H-MGD** | **H-LE** |
| 2:2 | AVG | 0.57 | 0.68 | 0.66 | 0.67 | 2037 | 88971 | 3.45 | 17.35 |
| | MAX | 0.79 | 0.82 | 0.87 | 0.83 | 3041 | 70810 | 4.72 | 11.30 |
| | MIN | 0.24 | 0.37 | 0.46 | 0.51 | 878 | 46410 | 3.54 | 11.66 |
| 3:3 | AVG | 0.54 | 0.72 | 0.62 | 0.61 | 7428 | 152219 | 9.35 | 21.86 |
| | MAX | 0.76 | 0.87 | 0.96 | 0.91 | 5172 | 159577 | 9.61 | 17.79 |
| | MIN | 0.20 | 0.25 | 0.30 | 0.32 | 1239 | 65195 | 8.16 | 17.56 |
| 5:5 | AVG | 0.64 | 0.75 | 0.60 | 0.61 | 11744 | 427050 | 31.63 | 38.67 |
| | MAX | 0.88 | 0.88 | 0.95 | 0.93 | 18272 | 383108 | 22.04 | 26.13 |
| | MIN | 0.13 | 0.20 | 0.20 | 0.17 | 1128 | 82743 | 17.32 | 25.12 |

Table 6.5: Results for Biomine

## 6.3 AVG, MAX, MIN: varying hops from center

In this series of experiments, we increment the distance between the source and the destination nodes for a single dataset and measure the effects on the reliability and execution time.

### 6.3.1 Experiment Setup

We gradually increment the radius of the node selection area from 2 to 4. The radius is relative to a center node, thus the maximum distance between any node pair is gradually incremented from 4 to 8 (see Appendix A.2). We are bound to use the Flixster dataset despite its bias against the heuristic algorithms, otherwise the experiment duration would be too long.

| | | Reliability | | | | Time (sec) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **BL1** | **BL2** | **H-MGD** | **H-LE** | **BL1** | **BL2** | **H-MGD** | **H-LE** |
| 2:2 | AVG | 0.40 | 0.51 | 0.43 | 0.43 | 9.02 | 3460 | 7.80 | 0.38 |
| | MAX | 0.66 | 0.79 | 0.71 | 0.70 | 9.12 | 3148 | 12.09 | 0.37 |
| | MIN | 0.15 | 0.28 | 0.15 | 0.17 | 4.30 | 4750 | 6.85 | 0.32 |
| 3:3 | AVG | 0.39 | 0.52 | 0.42 | 0.44 | 18.60 | 8569 | 17.55 | 0.65 |
| | MAX | 0.80 | 0.89 | 0.84 | 0.85 | 18.57 | 9043 | 19.73 | 0.65 |
| | MIN | 0.06 | 0.17 | 0.06 | 0.07 | 4.22 | 7599 | 8.19 | 0.62 |
| 5:5 | AVG | 0.46 | 0.55 | 0.46 | 0.48 | 100.52 | 26042 | 58.17 | 1.70 |
| | MAX | 0.95 | 0.98 | 0.96 | 0.95 | 63.01 | 25903 | 54.16 | 1.08 |
| | MIN | 0.02 | 0.12 | 0.05 | 0.04 | 7.37 | 22560 | 9.49 | 0.86 |

Table 6.6: Results for Flixster

**Experiment parameters**:

| | |
|---|---|
| Datasets: | Flixster |
| Number of colors: | $k$ = 5 |
| Number of shortest paths: | $r$ = 20 |
| MC-samples in Baseline 1: | $K$ = 100 |
| MC-samples in Baseline 2: | $K$ = 1000 |
| Source/destination nodes: | 2:2, 3:3, 5:5 |
| Number of queries: | 100 |

### 6.3.2 Results

Tables 6.7, 6.8 and 6.9 show the results for the various hops. We don't go too much into details as the values are all quite similar to the Flixster results from the previous experiment in Table 6.6. In general, the reliability goes down as we increase the number of hops, which is rather not surprising. The execution time remains on the same level for all algorithms regardless the hops. MIN-reliability for the heuristics is rather poor compared to Baseline 2. In the case of 2-hop 2:2, the heuristics fall even behind Baseline 1.

| | | Reliability | | | | Time (sec) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **BL1** | **BL2** | **H-MGD** | **H-LE** | **BL1** | **BL2** | **H-MGD** | **H-LE** |
| 2:2 | AVG | 0.39 | 0.50 | 0.41 | 0.39 | 7.79 | 4338 | 6.63 | 0.33 |
| | MAX | 0.62 | 0.77 | 0.69 | 0.67 | 7.72 | 3805 | 6.52 | 0.30 |
| | MIN | 0.16 | 0.29 | 0.14 | 0.13 | 3.12 | 3421 | 3.43 | 0.26 |
| 3:3 | AVG | 0.39 | 0.50 | 0.42 | 0.43 | 15.14 | 10117 | 14.88 | 0.47 |
| | MAX | 0.80 | 0.89 | 0.86 | 0.85 | 15.28 | 10090 | 14.95 | 0.45 |
| | MIN | 0.06 | 0.16 | 0.06 | 0.08 | 2.67 | 6846 | 4.77 | 0.38 |
| 5:5 | AVG | 0.39 | 0.49 | 0.42 | 0.43 | 47.68 | 26468 | 41.15 | 1.01 |
| | MAX | 0.94 | 0.98 | 0.96 | 0.95 | 52.56 | 26267 | 51.43 | 0.97 |
| | MIN | 0.02 | 0.06 | 0.03 | 0.02 | 3.45 | 17971 | 6.07 | 0.72 |

Table 6.7: Results for Flixster 2-hop around center

| | | Reliability | | | | Time (sec) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **BL1** | **BL2** | **H-MGD** | **H-LE** | **BL1** | **BL2** | **H-MGD** | **H-LE** |
| 2:2 | AVG | 0.32 | 0.45 | 0.37 | 0.35 | 5.71 | 4449 | 6.76 | 0.32 |
| | MAX | 0.63 | 0.77 | 0.69 | 0.67 | 5.70 | 3711 | 6.74 | 0.30 |
| | MIN | 0.08 | 0.21 | 0.09 | 0.08 | 1.74 | 3430 | 3.19 | 0.27 |
| 3:3 | AVG | 0.34 | 0.46 | 0.38 | 0.38 | 13.84 | 9411 | 15.41 | 0.47 |
| | MAX | 0.74 | 0.88 | 0.81 | 0.79 | 13.90 | 8747 | 15.26 | 0.46 |
| | MIN | 0.04 | 0.14 | 0.05 | 0.04 | 2.11 | 7239 | 4.07 | 0.40 |
| 5:5 | AVG | 0.38 | 0.47 | 0.40 | 0.41 | 43.17 | 30495 | 44.34 | 0.92 |
| | MAX | 0.94 | 0.97 | 0.96 | 0.96 | 47.82 | 31197 | 55.95 | 0.93 |
| | MIN | 0.01 | 0.04 | 0.01 | 0.01 | 1.91 | 15317 | 5.10 | 0.74 |

Table 6.8: Results for Flixster 3-hop around center

| | | Reliability | | | | Time (sec) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **BL1** | **BL2** | **H-MGD** | **H-LE** | **BL1** | **BL2** | **H-MGD** | **H-LE** |
| 2:2 | AVG | 0.25 | 0.36 | 0.29 | 0.27 | 4.78 | 4361 | 12.68 | 0.33 |
| | MAX | 0.53 | 0.72 | 0.59 | 0.57 | 4.82 | 4170 | 11.94 | 0.38 |
| | MIN | 0.06 | 0.14 | 0.06 | 0.05 | 1.45 | 3538 | 4.55 | 0.31 |
| 3:3 | AVG | 0.26 | 0.33 | 0.27 | 0.28 | 18.44 | 10134 | 26.61 | 0.51 |
| | MAX | 0.68 | 0.82 | 0.71 | 0.74 | 15.28 | 9293 | 25.44 | 0.49 |
| | MIN | 0.01 | 0.05 | 0.02 | 0.02 | 2.11 | 5389 | 4.84 | 0.39 |
| 5:5 | AVG | 0.22 | 0.31 | 0.25 | 0.25 | 37.41 | 28096 | 77.74 | 0.99 |
| | MAX | 0.80 | 0.91 | 0.84 | 0.85 | 37.36 | 25931 | 70.87 | 0.96 |
| | MIN | 0.00 | 0.01 | 0.00 | 0.00 | 1.65 | 10188 | 4.52 | 0.65 |

Table 6.9: Results for Flixster 4-hop around center

### 6.3.3 Analysis

To understand why MIN performs so badly, we analyze the returned colors for each individual query: besides 2:2 on 2-hop, H-LE doesn't find a single color on any query, which means the values we measured match the *random base reliability* for the particular query sets. The same is true for H-MGD and Baseline 1, although Baseline 1 manages to find some additional colors for 2:2 on 3- and 4-hops.

As we know from the single experiments, H-LE returns colors for only 40% of the queries. In 2:2 we have 4 pairs, assuming the same hit-rate, the probability to find colors for all 4 pairs would only be 3% and virtually zero for 5:5 queries, which correlates with the measured data in Table 6.7.

**Varying top-$k$**
We know from the single-source experiment analysis that shortest paths in Flixster have rather a lot of colors and that $k = 5$ is often too low to use them. To reduce this limitation we increase the number of colors $k$ and study the influence on the reliability in Table 6.10. We see that even with a larger $k$, H-LE is still on the same level as Baseline 1.
Unlike Baseline 1, and despite its poor reliability, H-LE finds colors for significantly more queries (43% of 100 queries, $k$=8) as with $k$=5 (3%). Due to the high *random base reliability* for eight colors, H-LE still cannot outperform Baseline 1.

| | | **Reliability** | | | **Time (sec)** | | |
|---|---|---|---|---|---|---|---|
| | Top-$k$ | **BL1** | **BL2** | **H-LE** | **BL1** | **BL2** | **H-LE** |
| 1-hop | 7 | 0.26 | 0.36 | 0.28 | 3.77 | 6085 | 0.28 |
| | 8 | 0.33 | 0.38 | 0.33 | 3.77 | 5459 | 0.27 |
| 2-hop | 7 | 0.28 | 0.37 | 0.31 | 2.74 | 5103 | 0.27 |
| | 8 | 0.34 | 0.40 | 0.36 | 2.89 | 5345 | 0.29 |
| 3-hop | 7 | 0.19 | 0.26 | 0.20 | 1.64 | 7613 | 0.27 |
| | 8 | 0.23 | 0.29 | 0.24 | 1.55 | 8613 | 0.28 |

Table 6.10: Varying top-$k$ on MIN for Flixster 2:2

**Case study**
To complete the analysis we look at one particular query with 2:2 source/destination nodes and $k = 8$ where MIN doesn't find any colors. The color sets for all of the shortest paths of each pair are shown in Table 6.11.

| Top-$r$ | **Pair 1** | **Pair 2** | **Pair 3** | **Pair 4** |
|---|---|---|---|---|
| 1 | {2,4,7,10,3} | {1,2,3,9} | {2,3,4,7,10} | {1,4,6,7,8,9,10} |
| 2 | {2,4,7,10} | {1,2,3,9,10} | {2,3,4,7,10} | {1,4,6,7,8,9,10} |
| 3 | {2,4,7,10,3} | | {2,3,4,7,10} | |
| 4 | {2,4,7,10,3} | | {2,3,4,7,10} | |

Table 6.11: Case study MIN paths ($k = 8$)

We only included the paths that have no cycles. We see that the color sets are similar, thus the MIN algorithm has not many possibilities to find matching sets. Indeed, there is no way to select a path from each pair such that the total number of colors is less or equal 8, even when the individual paths have less than 8 colors. This situation is rather common for higher $k$ although there are still some paths (even shortest paths) that exceed the color limit.

## 6.4 Other experiments

One other experiment was *Varying top-$r$ on Flixster*, similar to the top-$r$ experiment in Section 5.4.3 but with AVG, MAX and MIN. The outcome was the same as before, increasing $r$ would not improve the reliability. From the experiment for single sources we already know that increasing $r$ does not bring significant improvement in reliability. Given that the pairwise aggregations rely on their single $s$-$t$-paths, it is clear that these can't improve either.

# 7 Conclusion

In this thesis we presented the problem of top-$k$ colors in uncertain graphs for multiple-source/destination queries. We specified four possible optimization problems: Max-Average, Max-Maximum, Max-Minimum and Max-Connectivity. For the first three of them, the pairwise aggregations, we designed new baseline and heuristic algorithms based on the single-source/destination algorithms. For the fourth one, the connectivity problem, we adopted the existing approach to use minimal Steiner trees instead of shortest paths. We also proposed several optimizations to the basic algorithms that helped us to improve the run time performance of the heuristic solutions.

By running extensive experiments we demonstrated the performance of our baseline and heuristic solutions. We showed that the execution time of the heuristic solution outperforms Baseline 2 by several orders of a magnitude and beats Baseline 1 in terms of reliability.

In the experiment analysis we discovered some inherent weaknesses in the shortest path approach. Many of the retrieved paths other than the first, cannot be used for path inclusion, reducing the effective number of paths for two of our for datasets to less than two, regardless of the value of $r$. This is mainly caused by two issues with the paths: they may contain cycles or they may contain too many colors. These issues affect the pairwise aggregated multi-source problems in different ways. *Max-Maximum* is indirectly affected as the individual node-pair colors might be not optimal. *Max-Average* and *Max-Minimum* are directly affected as paths with many colors do not leave much range for the path inclusion algorithm to consider every node pair adequately, especially when the color limit is tight.

We see that H-MGD, while conceptually inferior, can compete with H-LE for close node pairs. Another interesting finding is that one intentional flaw in H-MGD, preferring short distance over short weighted paths, is also the reason why it outperforms H-LE on Flixster.

## 7.1 Future Work

In the experiments we have identified some issues with the *Most-Reliable-Path based heuristic* algorithm. In future work one could address these issues by improving the usefulness of the paths that are retrieved. The first point, cycled paths, could be solved by using a $K$ shortest path algorithm that doesn't allow cycles, like Yen's algorithm [YEN71]. As for the second point, paths with too many colors, one possible solution is to use the shortest path in terms of hops between the source and destination, to reduce the average number of colors in a path. One could also think of some combinations of low-hops and low-length paths. Fixing these issues would not only improve the reliability for single-source queries but also for the pairwise multi-source problems.

# A
# Datasets and Queries

## A.1 Datasets

This section describes the four datasets we use in our experiments.

### A.1.1 Freebase

"Freebase is large online collaborative online knowledge base" (Wikipedia[1]). The content is added and maintained mostly by voluntary community members. Freebase contains data from a wide range of other data sources like FMD, MusicBrainz and Wikipedia. Freebase content is organized in topics (vertices) and facts (edges).

The data graph itself doesn't provide edges with probabilities. Therefore uniformly distributed random values in the range of $[0, 1]$ were assigned to the edges.

Freebase is by far the largest dataset we use in the experiments, it has also the largest set of edge-colors. The average degree of edges on a node is lower than in the other datasets. However, the degree varies highly, many nodes have just one neighbor, while there are some *hub*-like nodes which may have hundreds of adjacent neighbors. The graph is partitioned into one large and several smaller sub graph that are not connected to each other.

[1] https://en.wikipedia.org/wiki/Freebase, 2015-07-15

Freebase properties:

| | |
|---|---|
| *Graph type:* | undirected |
| *Nodes:* | 28'483'132 |
| *Edges:* | 46'708'421 |
| *Avg. node degree:* | 3.3 |
| *Colors:* | 5428 |
| *Avg. colors per edge:* | 1 |
| *Edge probabilities:* | mean: 0.50, median: 0.50 |

### A.1.2 Biomine

Biomine is a biological graph database built-up from public databases. The vertices represent biological concepts such as genes, proteins, tissues, etc. The edges correspond to real word phenomena such as "a gene codes for a protein" or "an article refers to a phenotype" between those biological concepts. (Biomine Project[2])

Biomine is a medium sized graph with highly interconnected nodes.

Biomine properties:

| | |
|---|---|
| *Graph type:* | undirected |
| *Nodes:* | 1'045'414 |
| *Edges:* | 6'742'939 |
| *Avg. node degree:* | 12.9 |
| *Colors:* | 20 |
| *Avg. colors per edge:* | 1 |
| *Edge probabilities:* | mean: 0.27, median: 0.22 |

### A.1.3 Flixster

Flixster[3] is a social movie web site where users can discover new movies, learn about movies and find other users with a similar taste in movies. The vertices in the graph represent its users. An edge between two users means the source vertex (user) is friend with the target user (vertex) for a specific topic.

Flixster is the smallest dataset and the only directed graph. Each edge can have up to 10 topics (colors). To avoid multi-colored edges, the graph is converted into a multigraph with a separated edge for each color. A multigraph is a graph where two nodes can be connected by tow or more parallel edges. A characteristic of Flixster is its many edges with a probability of 1 or close to 1.

[2] https://www.cs.helsinki.fi/group/biomine/, 2015-07-15
[3] https://www.flixster.com/

Flixster properties:

| | |
|---|---|
| *Graph type:* | directed, multigraph |
| *Nodes:* | 29'357 |
| *Edges:* | 425'228 |
| *Avg. node degree:* | 29 |
| *Colors:* | 10 |
| *Avg. colors per edge:* | 4 |
| *Edge probabilities:* | mean: 0.20, median: 0.08 |

### A.1.4 DBLP

DBLP is a bibliography database focusing on computer science. It is hosted at the University of Trier, Germany. We process the data to build a graph. In this graph the authors are represented as vertex, an edge represents a topic of a paper that has both adjacent nodes listed as authors.

The topics are extracted from the paper title and matched against a list of predefined keywords. To calculate the cost of an edge, we count how many papers with this topic exist that have both persons as co-authors. The cost is normalized to a range between 0 and 1 using the formula $e_p = 1 - e^{-c/\mu}$ where $c$ is the number of common topics and $\mu = 5$ is a constant value.

DBLP is medium sized with a rather large number of different colors. Like in Flixster, two authors can be connected by multiple topics.

DBLP properties:

| | |
|---|---|
| *Graph type:* | undirected, multigraph |
| *Nodes:* | 1'251'335 |
| *Edges:* | 12'789'787 |
| *Avg. node degree:* | 20 |
| *Colors:* | 343 |
| *Avg. colors per edge:* | 2.4 |
| *Edge probabilities:* | mean: 0.22, median: 0.18 |

## A.2 Query generation

For our experiments, we generate 3 different types of query sets for single- and multi-source/destinations usage.

**Connected random**
Single-source/destination queries based on random node selection. We want to be sure there is an actual connection between the source and the destination, so we begin by finding all connected components of the graph. The source node is selected randomly among all nodes, for the destination we only consider nodes in the same connected component. We do not allow pairs where the source/destination is the same node. The distance between a selected pair of nodes can be as much as the diameter of the connected component.

**$n$-hop**
The $n$-hop method selects single-source/destination pairs where the distance between the two nodes is exactly $n$ hops. We select the source node randomly over the whole graph. Then we use a BFS search to collect all nodes with a hop distance of $n$ from the start. Finally we randomly select the destination from these nodes.

**$n$-hop around center**
Multi-source/destination queries with a guaranteed maximal distance between any selected node. We start by randomly select a center node $c$ (Figure A.1) followed by a BFS search to get all nodes with a distance of $n$ or less from the center node. The source and destination nodes ($s_1$, $s_2$, $t_1$, $t_2$) are then randomly selected among them. We do not select the same node twice. Pairwise distance between any selected nodes can be $2n$ or less.

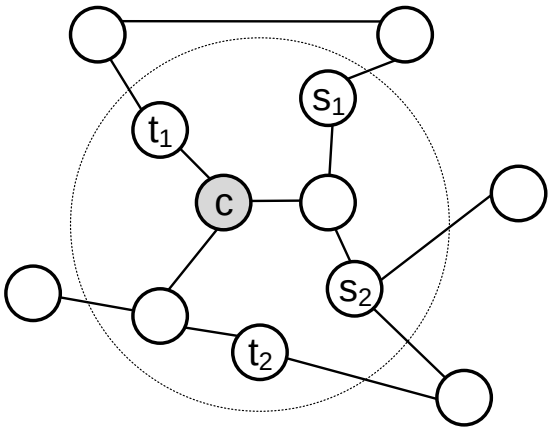



Figure A.1: 2-hop around center

# List of Figures

# List of Tables

# Bibliography

[KHAN15] A. Khan, F. Gullo, T. Wohler, F. Bonchi. Top-k Reliable Edge Colors in Uncertain Graphs. *In CIKM 2015*

[BALL86] M. O. Ball. Computational Complexity of Network Reliability Analysis: An Overview. *IEEE Tran. on Reliability, 1986.*

[FISH86] G. S. Fishman. A Comparison of Four Monte Carlo Methods for Estimating the Probability of s-t Connectedness. IEEE Tran. Rel., 1986.

[JIN11] R. Jin, L. Liu, B. Ding, and H. Wang. Distance-Constraint Reachability Computation in Uncertain Graphs. *PVLDB, 2011.*

[CHEN14] M. Chen, Y. Gu, Y. Bao, and G. Yu. Label and Distance-Constraint Reachability Queries in Uncertain Graphs. *In DASFAA, 2014.*

[SEVON06] P. Sevon, L. Eronen, P. Hintsanen, K. Kulovesi, and H. Toivonen. Link Discovery in Graphs Derived from Biological Databases. *In DILS, 2006.*

[EPP98] D. Eppstein. Finding the k-Shortest Paths. *SIAM J. Comput, 28(2):652–673, 1998.*

[JIME03] Víctor M. Jiménez and Andrés Marzal. A Lazy Version of Eppstein's K-Shortest Paths Algorithm. *In WEA Proc. on algorithms:179-191, 2003.*

[DING07] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, X. Lin. Finding Top-k Min-Cost Connected Trees in Databases. *In IEEE ICDE 2007.*

[YEN71] J. Y Yen. Finding the K-Shortest Loopless Paths in a Network. *Management Science 1971; 17:712–716*

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

## Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

---

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

**Title of work** (in block letters):

Top-k Color Set in Uncertain Graphs

**Authored by** (in block letters):

*For papers written by groups the names of all authors are required.*

| **Name(s):** | **First name(s):** |
|---|---|
| Nufer | Andreas |
| | |
| | |
| | |
| | |

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

| **Place. date** | **Signature(s)** |
|---|---|
| Zürich, 2015-09-11 | A. Nufer |
| | |
| | |
| | |
| | |

*For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.*