Calculating Upstream Relation in Spatial Networks Under Path Constraints

Wejdene Mansour
Professorship of Big Geospatial Data Management
Technical University of Munich, Germany
wejdene.mansour@tum.de

Martin Werner
Professorship of Big Geospatial Data Management
Technical University of Munich, Germany
martin.werner@tum.de

Abstract

Assessing the resilience of utility networks and ensuring human safety in indoor environments during emergencies relies on the identification of the upstream relation in a spatial network. This concept is essential for detecting vulnerabilities and maintaining operational continuity in the event of unforeseen disruptions by finding alternative paths under various constraints and accessibility restrictions. As spatial networks grow increasingly complex, it is no longer practical to comprehensively analyze all paths within a spatial semantic graph, e.g., in applications related to geographic information systems and navigation services. This paper presents a novel computational method–different from the conventional biconnected components–for analyzing spatial networks independent of any number of constraints. The defined approach employs a fast graph search algorithm based on shortest path queries with increasing graph weights. It integrates the constrained shortest path search with weighted edges and a parallelized linear search scalable to large networks. Unlike Block-Cut trees, our algorithm efficiently determines the upstream relation with directly embedded path constraints by identifying all simple paths that fulfill the specified constraints without entirely reconstructing the graph. We demonstrate our solution's efficiency across two infrastructure networks: a critical utility network and an indoor navigation map with different path constraints. The source code is available at https://github.com/tum-bgd/2024-sigspatial-upstream

CCS Concepts

• **Theory of computation** → **Graph algorithms analysis**; • **Mathematics of computing** → **Graph theory**; • **Computing methodologies** → *Parallel algorithms.*

Keywords

Spatial Networks, Graph Analysis, Path Constraints, Shortest Path

ACM Reference Format:
Wejdene Mansour and Martin Werner. 2024. Calculating Upstream Relation in Spatial Networks Under Path Constraints. In *The 32nd ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '24), October 29-November 1, 2024, Atlanta, GA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3678717.3691288

1 Introduction

The analysis of spatial networks has a long history and covers a variety of networks that are vital to the functioning of our modern society [2]. Nowadays, we have access to a large variety of such spatial graph datasets from diverse fields and with a wide range of sizes and complexities [22]. Especially interesting spatial networks are given by utility networks [15], including gas, water, and electricity supply networks, along with mobility networks, like road networks, train networks, and multimodal transport systems [19, 23]. Beyond traditional infrastructure, emerging location-based social networks and mobile communication networks provide a georeferenced network structure with intriguing properties [16]. In addition, spatial networks have been derived from popular video game maps such as StarCraft II [4], as well as floor plan maps of airports, office buildings, and university campuses [12, 31]. The latter types of graphs pose additional challenges for spatial analysis as their complexities are highlighted in their non-standard features, such as the presence of teleporters, elevators, multi-level structures, or restricted areas, affecting their connectivity and navigation.

Many of these networks provide services essential to the well-being of our societies and can be seen as examples of critical infrastructure—key services vital to the functioning of a country and its society [13, 17, 30]. Indoor maps of public spaces like airports and commercial buildings are characterized by high traffic, and the safety and security of visitors must be ensured at all times. These maps must be constantly evaluated for emergency response planning and efficient evacuation routes [24, 32], which can be effectively done using spatial graph analysis.

The impact of changes to a network on its overall functionality is often called resilience. However, the increasing flexibility and complexity of spatial networks means that it is no longer possible to understand their underlying connectivity. This hinders the assessment of the network's resilience and the development of concrete proposals to improve it. Due to their central meaning, it is increasingly important to analyze such spatial networks with the help of computational methods.

Since the components of utility networks fail on a regular basis, we must ensure that their overall design prevents these local failures from impacting the entire network in an uncontrolled manner. One significant example of badly designed resilience, which could have been routinely checked with upstream analysis, was the power outage in Berlin Köpenick. This incident resulted in more than 30, 000 households being disconnected from the electricity network for about 30 hours [18]. In this scenario, both the main cable and the backup cable were accidentally destroyed at the same time during construction work. The cause behind this is that both cables had to cross the same bridge—a problem for network resilience.

Building information modeling plays a critical role during the design phase and pre-construction stages of buildings to ensure that all security measures are adequately addressed. The tragic incidents in India and Bangladesh that resulted in numerous fatalities due to the absence of fire exits in large indoor spaces demonstrate the importance of proper planning of emergency evacuations relative to building size and its occupancy [1, 3]. Additionally, the many security lapses in airports, where restricted access areas were bypassed by passengers, have led to multiple breaches [25], prompting many airports to implement access control systems [29]. These examples highlight the importance of ensuring that indoor floor plans comply with safety requirements, guarantee operational efficiency, and eliminate security vulnerabilities.

One useful computational method for this type of analysis is the so-called **upstream relation**. To compute this relation, we first identify two sets of features in the network: the starting points and the controllers. The upstream relation is defined as the set of features of the graph, i.e., edges and vertices, for which a simple path between a start and an end feature exists. In other words, we are interested in finding the set of simple paths that connect starting points with controllers inside the network. These simple paths consist of network elements represented as vertices or edges in the graph. In a real-world example, this resembles the subnetworks that can be used to transport resources (energy, gas, water, etc.) between any consumer (starting point) and supplier (controller).

Given the upstream relation of a network, one can analyze how versatile and resilient an infrastructure is in case of faults. For the Berlin outage example, the controller would have been all available power plants within a sensible distance, and the starting points would have been any place inside Köpenick, for example, the town hall. While the upstream relation would have included many redundant paths to support this city, it would have also shown a bottleneck near the bridge.

On the other hand, the upstream relation is very interesting in the context of building floor plan design. Indeed, path constraints are highly relevant in real-world applications like airport management, where access points have certain accessibility restrictions. In this case, the upstream relation can help optimize passenger and airport personnel flow, improve emergency response planning to ensure visitor safety and comply with security protocols determined by individual security clearances. For instance, one can easily verify that newly integrated access control constraints do not hinder the workflow of an airport, as well as ensure compliance with emergency response protocols.

In 2018, the 26th ACM SIGSPATIAL conference highlighted this problem through its annual algorithm competition, GIS Cup, focusing on the challenge of identifying upstream features in large spatial networks [21]. The motivation behind the task was to analyze the electrical network of Naperville with respect to the upstream relation to help locate critical infrastructure assets. All three winning algorithms [11, 20, 28] in this competition used the concept of biconnected components to quickly infer the upstream, which was mainly motivated by its adequacy and computational efficiency.

This paper presents a novel and reasonably quick approach to identifying upstream features in the network, given a set of controllers, a set of starting points, and path constraints. In addition, we explain how this foundational configuration can be extended to deal with edges as starting points as well as with the resolution of special cases and vertices in the upstream relation. In contrast to the winning algorithms, however, we propose to actually compute a set of simple paths, which allows us to include path constraints like capacity or path length in the analysis. This is not easily feasible after compressing the graph into a simpler structure called the Block-Cut tree (BC-tree). A BC-tree highlights key connectivity features extracted from the decomposition of a graph into its biconnected components and articulation points. Nonetheless, thanks to their computational efficiency, existing BC-tree approaches can be effectively applied to prune large networks prior to applying constraints.

The remainder of the paper is structured as follows: In the next section, we define the upstream relation problem, discuss the general idea of the BC-tree-based solution, and introduce relevant shortest path algorithms. In Section 3, we derive our algorithm based on shortest path queries and present efficient implementation strategies to minimize computational effort. This section concludes with an illustrative example of the iterative process to compute the upstream relation and details how path constraints are integrated. Section 4 covers methods for efficiently processing large constrained graphs, while Section 5 addresses the handling of special cases. Following this, we evaluate the performance of our algorithm in Section 6 and demonstrate its importance in handling constraints in Section 7 using a constrained navigation problem on a university campus. Finally, Section 8 concludes the paper.

2 Problem Definition and Background

2.1 The Upstream Relation

Let us revisit the concept of the upstream relation [21]. To formalize this, we begin by establishing the appropriate notation and defining the problem we seek to solve, namely querying the upstream relation of a tuple $(G(V,E), S, C)$:

Problem 1. *Given a directed or undirected graph $G(V,E)$ composed of a set of vertices V and edges E, a subset of vertices and edges serving as starting features of the search $S \subseteq (V \cup E)$, and another subset of vertices serving as controllers $C \subseteq V$, enumerate all upstream features, i.e., edges and vertices, with respect to the start features and controller vertices.*

To clarify the upstream relation, we first define what is considered an **upstream feature** in this context:

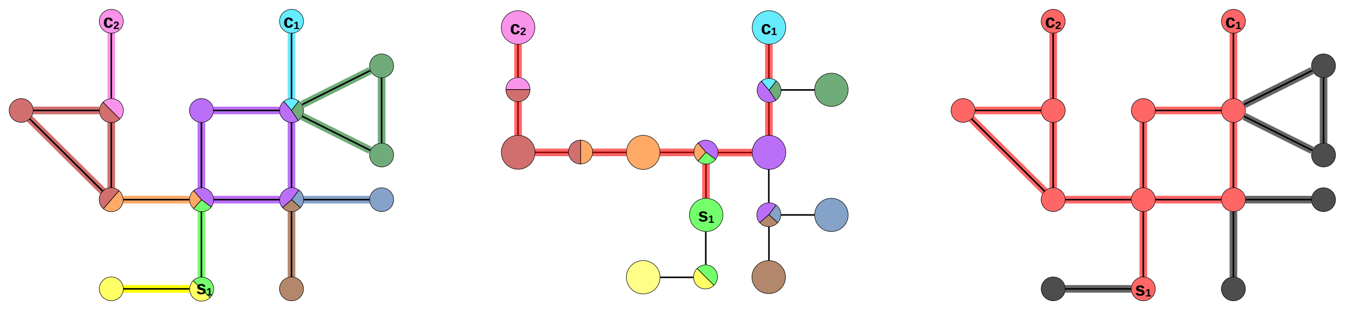
Definition 1. *A feature, edge or vertex, $f \in (V \cup E)$ is upstream if it is part of a simple path from any start feature $s \in S$ to any controller $c \in C$.*

For completeness and to better understand what constitutes a **simple path**:

Definition 2. *A path in a graph is set to be simple if it does not have self-intersections – that is, it does not contain a certain vertex more than once.*

2.2 BC-Tree Solution

This simple definition of upstream relation is closely related to the graph concept of biconnected components, which helps understand the connectivity and structure of the graph. The significance of this



(1) Graph decomposition into biconnected components **(2) Reduction of the components into the Block-Cut tree** **(3) Upstream features between the starting point and controllers highlighted in red**

Figure 1: Solution of the upstream relation in an example utility network between a starting point s_1 and two controllers c_1 and c_2. The upstream relation is computed based on the idea of biconnected components and Block-Cut tree.

concept can be clearly noted in the three winning GIS Cup submissions, each of which specifically implemented this idea [11, 20, 28] while integrating additional steps to enhance runtime performance.

An illustration demonstrating the effectiveness of this approach in solving this challenge is provided in Figure 1. A biconnected component denotes a connected subgraph such that there exist two distinct paths between any pair of vertices within this subgraph. In other words, the subgraph remains connected regardless of which single vertex we remove. For this reason, we must identify bridges and articulation points, or cut vertices, responsible for potentially disrupting the graph connectivity. Each biconnected component is highlighted in a different color in (1), with cut vertices having the colors of the different biconnected components they belong to.

Since any vertex within a biconnected component is always reachable without leaving the component, a biconnected component is either completely contained inside the upstream or entirely outside of it. Consequently, it makes sense to partition the graph into its biconnected components and reduce them to a single node, resulting in a compressed tree structure: the BC-tree. Given this BC-tree in (2), the upstream relation is straightforward to calculate due to the fact that trees inherently lack loops: all tree blocks along the path from the starting point block and the controller block are upstream. In the final step (3), we uncompress the BC-tree, and all features within an upstream tree block are also upstream.

The conventional algorithm for generating this tree is based on depth-first search and is known as Tarjan's algorithm [27], which runs in linear time. Parallel implementations of this algorithm exist, achieving sublinear time complexities [26], with significant speedups being achieved with parallel implementations [6].

However, a significant shortcoming of this method is the fact that path properties within biconnected components might be non-uniform. Consequently, when attempting to extend these algorithms to include constraints on the paths, the internal structure of each biconnected component must be evaluated again, and the benefit of first decomposing the graph into these components is potentially lost. Therefore, in our paper, we tackle this limitation by proposing to take a more direct approach. This is realized by implementing a structured search of shortest paths to identify simple paths, proving that a given feature is upstream.

2.3 Shortest Path Algorithms

Computing shortest paths is a longstanding computational challenge. In this context, several concrete problems must be distinguished: All-Pair Shortest Path, also known as Multiple Source Shortest Path (MSSP), Single Source Shortest Path (SSSP), and determining the shortest paths between two given points.

MSSP algorithms, such as the Floyd-Warshall algorithm [9], create a distance matrix between all nodes according to network distance. This distance matrix can be used to compute the shortest path between any two pairs of vertices using a sequence of table lookups.

Given a starting point, the SSSP problem computes the distance and shortest path to any other node. The most prominent example of such an algorithm is Dijkstra's algorithm [5], which serves as the foundation for all modern shortest-path algorithms. In order to not store all the paths that come up, it uses two properties assigned for each vertex: the distance and the predecessor vertex. A shortest path is then constructed by following the predecessor's stored at each vertex. From an algorithmic point of view, it is needed to show that the shortest paths are, in fact, shortest, and this is done by maintaining three sets of vertices often represented by colors. The white vertices represent nodes that have not yet been seen by the algorithm; the gray vertices denote nodes that have been visited at least once but for which the shortest path is yet to be discovered, i.e., nodes that are currently in the queue; and the black set of vertices for which we know that no shorter path can exist. Using a priority queue, the algorithm traverses the tree, filling in the three properties: color, distance, and predecessor.

In fact, the shortest path between two points often involves solving the all-pair shortest path problem for either one or both of the points in parallel. However, to optimize its efficiency, we refine the algorithm to avoid exploring irrelevant sections of the graph for which we can easily conclude that they will not contribute to the shortest path.

Among the notable variants of Dijkstra's algorithm is the A* (A star) algorithm [14], which uses the triangle inequality as a lower bound to the distance, thereby disregarding vertices that are too far away. Another significant approach, the A*, Landmarks and Triangle (ALT) algorithm [10], precomputes the distance map for select vertices, typically placed at the borders of the graph. By utilizing the triangle inequality as well, the ALT algorithm achieves a tighter lower bound than the A* algorithm. Additionally, contraction hierarchies add shortcuts to the graph based on several proposed schemata such that shortcuts can be used to jump over larger subgraphs occurring along a path. For example, in a scenario where a village has a single road entering and another road exiting, this configuration can be jumped over as if it was a single vertex. Furthermore, the nature of street networks is often exploited in such hierarchies. For example, we consider the Mississippi River, where only a handful of roads cross the river. In such cases, each coast-to-coast travel can be decomposed into a few shortest path computations on the two graph sections on either side of the river. In fact, a contraction hierarchy would finally start with identifying the correct bridge and then descending one level to refine the shortest path on both sides of the bridge.

Given that the upstream relation can be computed in any arbitrary graph, which might be dynamic, we propose to use the ALT search algorithm instead of contraction hierarchies. Indeed, in this case, these graphs are subject to changes not only in weights but also in topology. This technique is characterized by its fast performance and minimal preprocessing overhead, which consists of only a few Dijkstra searches, and it is compatible with increasing edge weights. Furthermore, we identify ideal landmark candidates for real-world upstream relations, namely the controllers and the starting points.

3 Enumerating Upstream Edges

The fundamental concept of this algorithm is to enumerate the paths that prove the upstream relation. Specifically, the basic algorithmic idea is to iterate over each edge of the graph and determine whether a simple path containing this edge exists. For ease of exposition, we focus solely on the case of upstream edges, as handling vertices involves a straightforward enumeration of special cases. In order to reach useful performance, we introduce efficient pruning strategies to avoid unnecessary computations.

3.1 Algorithm Derivation

Based on the definitions provided in Section 2.1, we want to find all edges part of a simple path between a starting feature and a controller. With this task in place, we can formulate the basic observation on which our algorithm is based as follows:

Lemma 1. *An edge $e = (p, q)$ is upstream if and only if there is a shortest path T from some starting point s to p (resp. q) and a shortest path R from the other point q (resp. p) to some controller c in G with all edges adjacent to T removed.*

Proof. Shortest paths are simple paths. Hence, both paths T and R are simple. T and R are disjoint as R is computed in the subgraph of G with all edges adjacent to T being removed. Therefore, the concatenation $[T, e, R]$ is simple as well. Conversely, let e be an edge and let T be a simple path from a starting point s via e to some controller c. Then, the paths $s \longrightarrow e$ and $e \longrightarrow c$ are disjoint. Hence, $e \longrightarrow c$ is a path in the subgraph $G \setminus \{s \longrightarrow e\}$. Consequently, the set of such paths is non-empty and, therefore, contains a smallest element, the shortest path. In summary, given a simple path, we can conclude that there is a pair of shortest paths, as shown in the lemma. □

While this observation specifically covers the edges case, we believe it represents the most intuitive formulation of how the upstream relation can be computed using shortest path queries on graph G and on certain subgraphs with only increased weights. For instance, setting the weights of adjacent edges to the path T to infinity is equivalent to removing the edge, but simplifies the computational processes as no updates to the adjacency list structure are required. Subsequently, as we elaborate later in the paper, the upstream features can be extracted from the problem record and the set of upstream edges.

This leads to a basic algorithm for the **decision problem** of whether a given edge is upstream with relation to a specific pair of a starting feature and a controller. Note that this method can be used in isolation; in other words, we can apply it independently to a single edge when examining the significance of a local edge in relation to a specified set of controllers and starting points.

Algorithm 1 The Decision Algorithm

```
Input: Graph $G$, Edge $e = (p, q)$, Starting point $s$, Controller $c$
Output: Upstream status of edge $e$
1: compute these combinations of shortest paths:
       $P_{1,1} : s \longrightarrow p$ in $G$   and   $P_{2,1} : q \longrightarrow c$ in $G \setminus \{s \longrightarrow p\}$   (1)
       $P_{1,2} : s \longrightarrow q$ in $G$   and   $P_{2,2} : p \longrightarrow c$ in $G \setminus \{s \longrightarrow q\}$   (2)
       $P_{1,3} : c \longrightarrow p$ in $G$   and   $P_{2,3} : q \longrightarrow s$ in $G \setminus \{c \longrightarrow p\}$   (3)
       $P_{1,4} : c \longrightarrow q$ in $G$   and   $P_{2,4} : p \longrightarrow s$ in $G \setminus \{c \longrightarrow q\}$   (4)
                 ▷ for $P_{2,*}$, set edges adjacent to $P_{1,*}$ to infinite weight
2: if $(P_{1,1} \wedge P_{2,1}) \vee (P_{1,2} \wedge P_{2,2}) \vee (P_{1,3} \wedge P_{2,3}) \vee (P_{1,4} \wedge P_{2,4})$ then
3:     return true                                  ▷ $e$ is upstream
4: else
5:     return false                                 ▷ $e$ is not upstream
```

This algorithm is employed in the upstream analysis process by iterating over all edges and invoking the decision algorithm.

Algorithm 2 Enumerating Upstream Edges

```
Input: Graph $G(V, E)$, Starting points $S$, Controllers $C$
Output: Property map on edges $E$ with upstream status
1: for $e \in E, s \in S, c \in C$ do
2:     invoke decision algorithm on $e$:
3:     if $isUpstream(G, e, s, c)$ then
4:         $upstream[e] \longleftarrow true$              ▷ $e$ is upstream
5:     else
6:         $upstream[e] \longleftarrow false$             ▷ $e$ is not upstream
7: return $upstream$
```

Algorithm 2 presents the baseline iterative process over all edges and is, therefore, neither optimized nor parallelized. Additionally, the proper handling of constraints must be further introduced.

3.2 Implementation Details

Each of these decisions is based on evaluating a shortest path predicate in the graph eight times, half of which get evaluated in the same graph with possibly higher edge weights. In addition, the set of possible end points of these graphs is exclusively limited to the union of S and C. Therefore, a landmark search engine [10] with the landmark set $S \cup C$ is a good fit for this problem as long as this set is reasonably small. If it is not, some of the various landmark selection schemes can be used in order to prepare only a few landmarks, but in most cases, the set will be small in practice.

In the case that the set is small enough such that every element of $S \cup C$ becomes a landmark, we can simplify the potential function of ALT to only use the goal distance from the distance map.

Implementation 1. *We implement the presented algorithm using a variant of landmark search in which each element of $S \cup C$ becomes a landmark and, consequently, the landmark search is most efficient for paths involving S and C. In addition, it returns the generated simple path composed of both shortest path segments for further processing.*

For the correctness of this algorithm, in general, we should remark that the algorithm uses a constant-zero landmark in cases where a shortest path is queried which does not start or end in $S \cup C$ and, thus, degenerates to a Dijkstra search. For the upstream relation, however, this cannot happen.

3.3 Optimizations and Parallelization

To speed up the processing of the main Algorithm 2, we use parallelization and early stopping, as presented in the following algorithm. By modeling the upstream status as a property map of edges, we can access (read and modify) the already calculated relation for an edge through an atomic operation and in constant time.

First, we initialize the search engine by running Dijkstra's Algorithm for $S \cup C$. Unfortunately, the decision algorithm is not thread-safe because it (a) modifies the graph weights and (b) relies on data structures specific to graph search. Next, we instantiate as many threads as the CPU sensibly supports and set up a thread-local copy of the search engine for each thread. For each edge, submit a decision problem as a task assigned to some of the available worker threads. Each thread gets a copy of the search engine, including the graph, the predecessor map storing the shortest path result, the distance map used for calculations, a priority map that accelerates graph search, and a set of non-traversable edges. Before invoking the decision algorithm aspects—the four combinations in Algorithm 1—for an edge e, we first check whether it has already been identified as upstream by the same or another thread.

We stop processing as soon as our algorithm has revealed that the given edge is upstream, which can happen in any of the four cases defined in Algorithm 1. Exceptions are used to exit the algorithm when necessary, with this exception being caught at the end of the decision task for this edge. Thereby, useless thread invocations are reduced to a Boolean comparison to check if the edge is upstream, resulting in an empty exception being thrown and caught. Thus, these invocations are handled very efficiently.

If a path exists, the decision algorithm is implemented so as to return the simple path through edge e. As a result, we significantly improve the performance by concluding that, in addition to edge e,

Algorithm 3 Parallelized Algorithm with Early Stopping

```
Input: Graph $G(V, E)$, Starting points $S$, Controllers $C$
Output: Property map on edges $E$ with upstream status
1:  $\mathcal{E} \leftarrow Dijkstra(G, S \cup C)$          ▷ initialize search engine
2:  instantiate $n$ threads
3:  create thread-local copies of $\mathcal{E}$
4:  for $e \in E, s \in S, c \in C$ do                    ▷ parallel processing
5:      submit decision problem for $e$ to available thread:
6:      try
7:          if $upstream[e]$ then
8:              throw $done$
9:          else
10:             invoke decision algorithm on $e$:
11:             if $isUpstream(\mathcal{E}, e, s, c)$ then
12:                 for $e' \in$ edges$(P)$ do           ▷ simple path P via $e$
13:                     $upstream[e'] \leftarrow true$
14:                 throw $done$               ▷ early exit decision algorithm
15:     catch $done$
16: return $upstream$
```

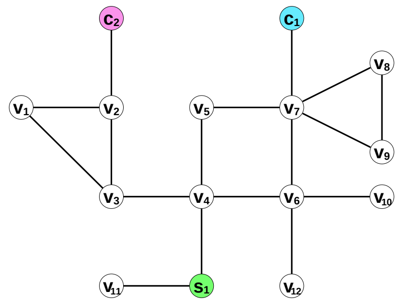
all edges within this path are upstream. We atomically update the global shared property map accordingly, and thus, no thread synchronization is required. Additionally, we note that it is sufficient to assess that an edge is upstream for a single (s, c) pair.

3.4 Illustrative Example

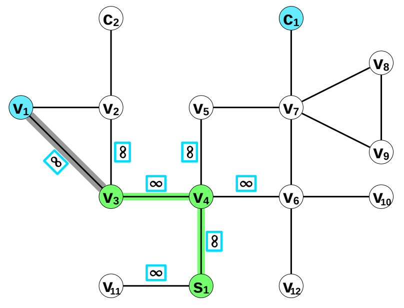
In Figure 2, we show the iterative process of the proposed algorithm on an example utility subnetwork. From the graph depicted in (1), we can identify one starting point s_1 and two controllers c_1 and c_2. Our algorithm employs Dijkstra's Algorithm with $S \cup C$ as landmarks to initialize the search engine. In the following steps, we iterate through all the edges in the graph, attempting to compute a simple path from the starting point s_1 and either of the controllers c_1 or c_2 and passing by the current edge.

We start with edge (v_1, v_3) and search for a simple path between s_1 and c_1 by looking at the four combinations of shortest paths defined in 1. An example of these four cases is shown in step (2), where even though we find a short path $s_1 \rightarrow v_3$, no short path can be found between v_1 and c_1 after setting all edges adjacent to $s_1 \rightarrow v_3$ to ∞. As a reminder, we set these adjacent edges to ∞ to make them non-traversable and enforce that no vertex is included more than once in the combination of the two subpaths, which is required to get a simple path. Since no simple path was found between s_1 and c_1 via (v_1, v_3), in step (3), we look into the same edge but with respect to the second controller. Indeed, we are able to find a simple path from the concatenation of two short subpaths, where the second path is computed in the same graph but with edges adjacent to the first one set to ∞: edge (v_1, v_3) is upstream. In step (4), we conclude that all edges along the simple path $s_1 \rightarrow v_3 \rightarrow v_1 \rightarrow c_2$ are upstream, and we no longer need to evaluate them with the decision algorithm. The same process is repeated for edge (v_2, v_3) as is shown in steps (5-7).

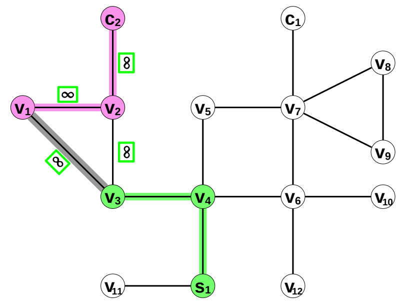
In steps (8-9) and (10-11), we directly identify a simple path from the starting point to the controller vertex c_1 via edges (v_4, v_5) and (v_4, v_6), respectively. All edges along these two simple paths are



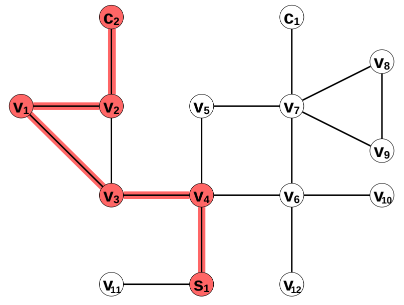
(1) Run Dijkstra's Algorithm for $S \cup C$ as landmarks to prepare the search engine.



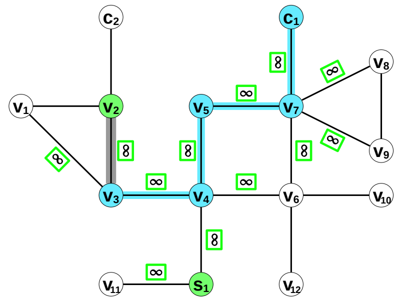
(2) (v_1, v_3): no simple path exists from v_1 to c_1 with edges adjacent to $s_1 \to v_3$ set to ∞.



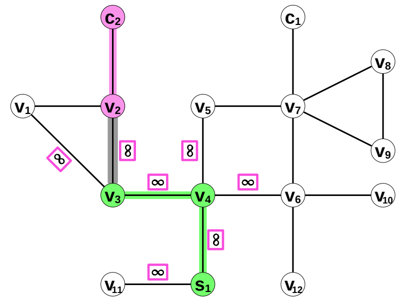
(3) A simple path exists between s_1 and c_2 through edge (v_1, v_3). (v_1, v_3) is upstream.



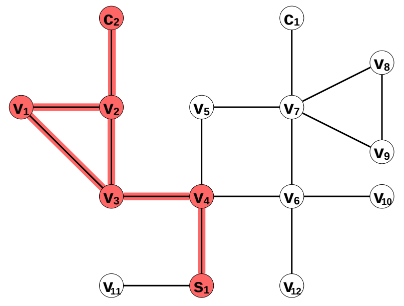
(4) Set all edges in the computed shortest path for edge (v_1, v_3) as upstream.



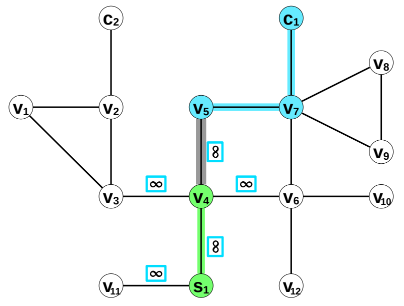
(5) (v_2, v_3): no simple path exists from v_2 to s_1 with edges adjacent to $c_1 \to v_3$ set to ∞.



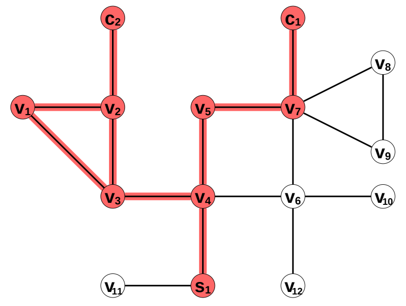
(6) A simple path exists between s_1 and c_2 through edge (v_2, v_3). (v_2, v_3) is upstream.



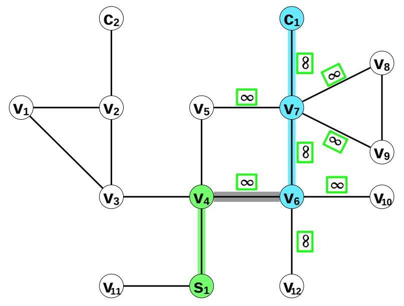
(7) Set all edges in the computed shortest path for edge (v_2, v_3) as upstream.



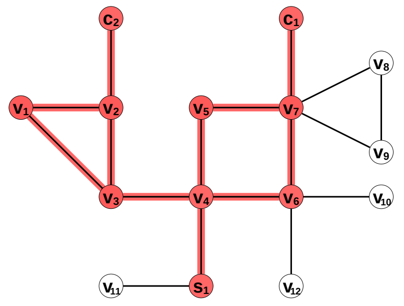
(8) A simple path exists between s_1 and c_1 through edge (v_4, v_5). (v_4, v_5) is upstream.



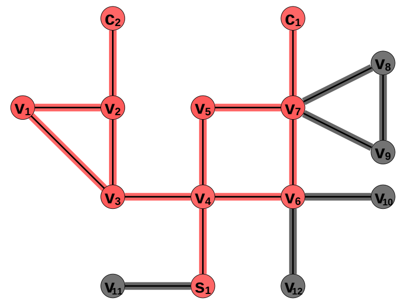
(9) Set all edges in the computed shortest path for edge (v_4, v_5) as upstream.



(10) A simple path exists between s_1 and c_1 through edge (v_4, v_6). (v_4, v_6) is upstream.



(11) Set all edges in the computed shortest path for edge (v_4, v_6) as upstream.



(12) No simple paths are found for the rest of the edges. Final upstream relation.

Figure 2: Solution of the upstream relation in an example utility network based on the proposed algorithm for upstream edge enumeration using shortest paths. We highlight the current edge of interest in gray, the upstream relation in red, and non-upstream features in black.

upstream, and besides, we do not need to evaluate the same edge for a simple path to the second controller since the edge is already upstream. For the rest of the edges that have not yet been visited and were not labeled upstream as part of a simple path for another edge, no simple path is found from the starting point to either controller. The final upstream relation with all upstream features highlighted in red is shown in step (12).

3.5 Path Constraints

For ease of exposition, we have explained how to compute the upstream relation from a sequence of shortest path queries or, more concretely, simple path existence queries. In very large networks, however, we will face a problem: the number of path computations scales linearly in the number of edges of the network.

In practical applications like utility networks, however, various constraints often limit the usability and efficiency of paths. For instance, the length of a useful path can be constrained by physical, logistical, and economic factors. Transporting goods or resources over excessively long distances is simply not possible or sensible due to potential delays and the increase in costs and energy consumption. For the semantic web example, short and contextually relevant connections are more meaningful than far-distance ones between any two nodes in the semantic graph and, thus, should be prioritized. In the case of an indoor navigation problem, constraints are even more common. Certain areas within a building might be inaccessible due to security restrictions, maintenance work, or safety precautions. In addition, paths within some environments might have certain properties that are needed for an application, including sufficient capacity to handle traffic, performance measures like time efficiency, and cost constraints related to resources.

These various path constraints can be seamlessly embedded into the presented framework by replacing the shortest path calculation with a constrained shortest path calculation, effectively deciding whether a simple path with the given constraints exists. In general, constraints only result in increasing edge weights, either gradually or instantly. For example, when a distance limit is reached, all outgoing edges can be assigned infinite weight to prune the search space. These methods are fully compatible with the landmark indexing scheme for shortest paths. We expect that in huge networks, e.g., transport networks and semantic web, a distance threshold is routinely set in order to limit the number of edges that can contribute to the upstream relation. This can be set absolutely, e.g., based on the transport cost or relative to the distance between any source and controller.

Unlike our proposed algorithm, the solutions presented for the challenge do not take this into consideration [11, 20, 28]. Indeed, we cannot apply distance or cost thresholds when using BC-tree. If we wish to explore various upstream relations with different restrictions, we would have to completely reconstruct the graph entirely and remove the features that do not align with these properties.

4 A Faster Variant for Large Networks

The algorithm described so far turns out to be fast, as we will detail later with reference to Esri's Naperville dataset example given on the challenge webpage. This efficiency is due to the size of the spatial network, which is still rather small, and the upstream relation being pretty local within this graph.

In this section, we propose two aspects that can be used to increase performance on very large networks: The first is biconnected pruning as proposed in [20], and the other is path penalization.

4.1 Biconnected Pruning

While we highlighted the challenges of reducing the problem to the BC-tree, which makes it difficult to impose constraints on the paths, it can still be used to prune components that cannot be upstream. Indeed, unlike the computationally efficient BC-tree solution, our algorithm, although fast, mainly focuses on enabling the integration of constraints into the upstream computation. To solve the upstream relation for large and constrained spatial networks, we propose to initially compute the BC-tree for pruning using the highly efficient solution of the challenge winners [11, 20, 28]. This exploits the fact that if a biconnected component is not upstream, it cannot become upstream under path constraints. We can then apply our method to the reduced unconstrained subgraph to remove the features that do not comply with the restrictions.

4.2 Path Penalization

In the case of large biconnected components, it can be beneficial to tweak the search of shortest paths in a way such that each path automatically explores another part of the graph. As long as no constraint is imposed on the length of the path, we can simply increase the weights of previously visited paths. Consequently, the probability is reduced that a future shortest path shares common subpaths with this path. This is connected to the fact that if we compute a single path, we will conclude for each of its edges that it is upstream. Therefore, discovering different paths will speed up the procedure.

5 Handling Upstream Vertices and Special Cases

While defining the upstream relation in terms of edges is intuitive, we wish to identify the set of all upstream features within a network, including both edges and vertices. This comprehensive approach unveils the entire set of network elements (transport connections, transformers, pumps, etc.) that are part of the upstream connections. However, despite its apparent simplicity, this task is not without complexities. Indeed, some specific cases require careful consideration, of which we highlight only the most interesting:

- If a starting point is an edge, both vertices can be upstream, one can be upstream, or neither is upstream.
- If a starting point coincides with a controller, we agree that this is upstream for practical purposes. However, it is not clearly upstream in the sense of Definition 1 because it is not part of any path at all.

For the case of starting edges, we propose to alter the graph by introducing a novel vertex s in the middle of the starting edge $s_e = (p, q)$ to serve as a start vertex, as shown on the left of Figure 3. The drawback of this adjustment is that it alters the graph structure. If this cannot be tolerated—where should this virtual vertex be located in the physical world?— it is possible to handle this algorithmically by involving a certain set of special cases. Another approach, depicted in Figure 3 on the right, is to disable the starting

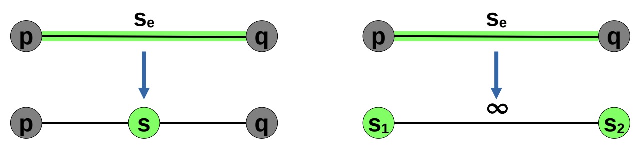


Figure 3: Two possible approaches to handle the case of a starting edge instead of a starting vertex: (left) introduce a ghost vertex or (right) disable the starting edge.

edge by assigning it an infinite weight. Hence, this edge is excluded from any potential path computations and instead the vertices on both ends are now the starting points. When determining the upstream status of the starting edge, if any of its adjacent edges are found upstream, the starting edge is considered upstream as well.

6 Implementation Performance

6.1 Naperville Electric Utility Network

To evaluate the algorithm, we first use the two datasets provided by the 2018 ACM SIGSPATIAL GIS Cup challenge organizers [21]. The first one is rather trivial, namely a small sample dataset with only 34 features to help verify the soundness of the proposed algorithm. The second dataset, Esri's Naperville Electric Utility Network, is a real-world electric network of the city of Naperville with a total of 17, 566 features, of which 8, 465 are vertices and 9, 101 are edges. We note that both datasets are represented by unweighted bidirectional graphs. Figure 4 illustrates a sample extract showcasing a subset of features from the Naperville electric dataset. The example of an upstream relation between a starting point in the northwest and a controller in the southwest is displayed in cyan.

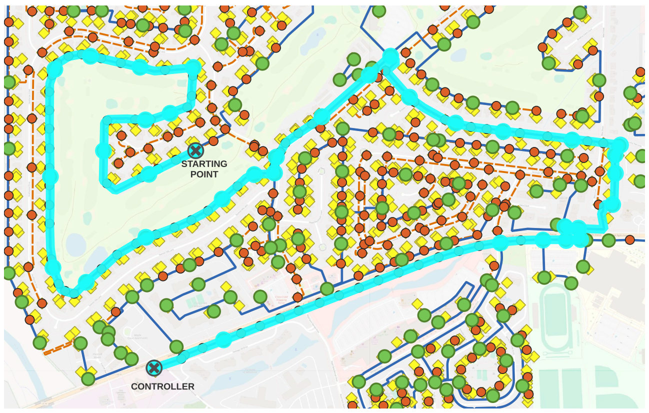


Figure 4: Extract from the Naperville Electric Dataset [21] with the upstream features of an example upstream relation between one starting point and one controller highlighted in cyan.

6.2 Performance

The performance tests presented in this paper are based on evaluations conducted on a workstation and averaged over 20 runs. The first dataset is a sample dataset that is too small to measure runtime performance. With a precision of milliseconds, the algorithm's execution time is measured to be 2.4ms for a parallelization of eight threads. This is the case due to the thread setup cost and the copying of the search engine. For the medium-sized dataset containing the electric network of the city of Naperville, we measure the execution times of the given instance on a single machine for varying numbers of threads.

Figure 5 depicts the performance of the main algorithm excluding I/O. It improves very quickly for small numbers of threads. This is due to the fact that the search engine indices created from Dijkstra's algorithm are getting parallelized. After that, we can see a linear trend of slowly decreasing times, reaching 289ms on 8 threads. This is the case where the decisions are being parallelized, but the number of threads is larger than the needed landmarks. Indeed, the graph search preparation and the treatment of the non-upstream edges dominate runtimes for higher numbers of threads.

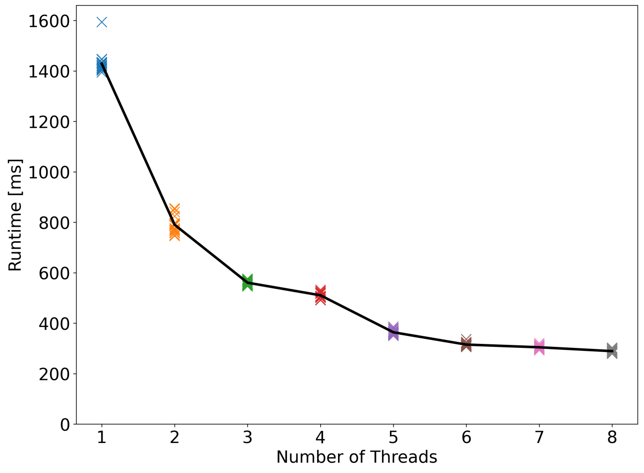


Figure 5: Performance scaling of the proposed algorithm with increasing number of threads on the Naperville dataset.

As we stated beforehand, unlike with the challenge, our algorithm does not achieve the best computational efficiency. Instead, our main focus is the efficient handling of constraints. With this first experiment focusing on runtime performance, our aim is to demonstrate that our introduced algorithm is still capable of quickly identifying the upstream relation. To handle even larger datasets with constraints, we suggest first reducing the graph to the subgraph of upstream relation without constraints using the existing BC-tree approaches. This is followed by applying our algorithm to filter out the paths that do not satisfy the constraints.

7 Constrained Upstream Relation

7.1 Complex Multistory Indoor Building

An additional type of dataset we want to analyze is a complex multistory indoor building. Such maps are especially interesting thanks to the various constraints they can be subjected to, which is the main focus of our proposed algorithm. To this end, we consider the floor plan of the Idaho State University Meridian campus [7, 8], which

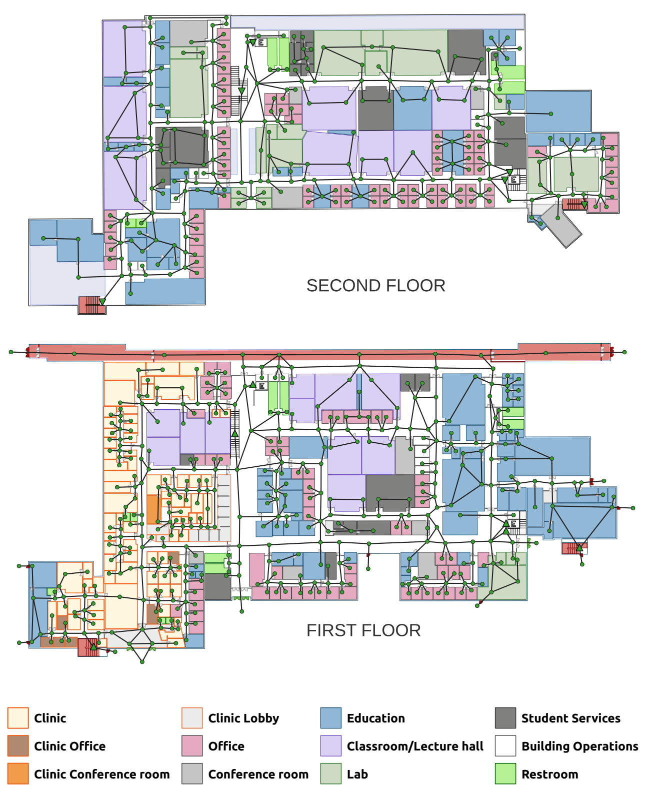


Figure 6: The floor plan map and graph network of a two-story university campus showcasing emergency exits and emergency exit hallways highlighted in red and rooms color-coded based on their functionality [7, 8]. The two floors are connected with multiple elevators and staircases.

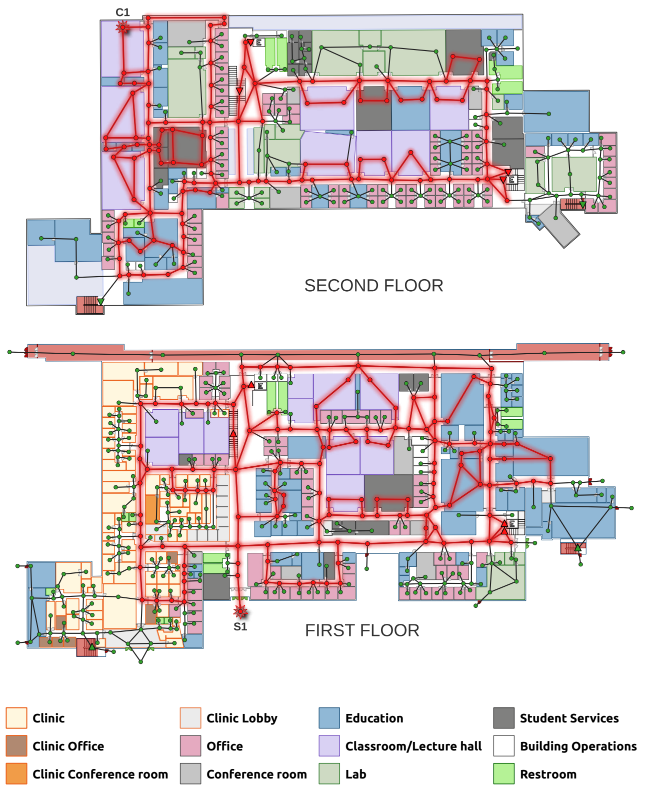


Figure 7: An upstream relation under constraints with the main entrance of a university campus as the starting point and a lecture hall on the second floor as the controller. To reflect the access restrictions to certain areas (clinics, clinic offices, offices, labs, and building operations), we apply absolute constraints, meaning the edges are considered non-traversable. Elevators and non-emergency staircases are used to move from one the first floor to the other.

we use to construct an undirected graph. We represent rooms and offices as nodes and connect them with constrained and weighted edges to reflect their distances and access restrictions. In contrast to the utility dataset, we select this indoor map to closely examine the importance of our algorithm in handling constraints that arise during navigation and emergency response evacuation. Indeed, as is shown in Figure 6, the university campus consists of two floors connected by escalators and elevators, with multiple emergency exits highlighted in red and color-coded rooms to indicate their function, which we use to implement accessibility limitations.

7.2 Spatial Analysis

To look into our algorithm's ability to handle constraints, we analyze two examples of constrained path searches on the indoor navigation map for efficient movement based on door accessibility restrictions and emergency exits under length constraints.

In the first example, our aim is to guide student flow in the university building to identify the main pathways while ensuring they bypass certain areas. This is an important step in preserving privacy and preventing unauthorized access while evaluating access and connectivity of the building. Moreover, we can use the upstream relation to analyze the shortest path for potential bottlenecks in narrow hallways or single points of access. In Figure 7, we use our algorithm on the generated graph and show the computed upstream relation between the main entrance on the first floor and an example lecture hall on the second floor. We assess the algorithm using absolute constraints derived from limiting access to clinics, clinic offices, offices, labs, and building operations. Furthermore, all emergency exits and emergency exit hallways must always remain unobstructed and are therefore off-limits. We can clearly see that none of the prohibited areas were part of the upstream relation, which is highlighted in red, meaning that our algorithm succeeded in complying with the specified restrictions. To get from the first level to the second, the nodes for the elevators and non-emergency staircases are visited.

As mentioned in the introduction, depending on the anticipated number of occupants in a building, the analysis of the redundancy of emergency exit routes is essential. This assessment ensures the availability of alternative pathways and prevents potential congestion points. During evacuation procedures, elevators and escalators are banned, and the entire emergency route must remain unobstructed and accessible to everyone without requiring special permission. Therefore, elevators are also prohibited in addition to maintaining the previously established absolute constraints.

Furthermore, evaluating the evacuation routes for both safety and efficiency is crucial to ensure rapid access to the emergency exits when necessary. Since our graph lacks travel time information between vertices, we treat the efficiency constraint as a restriction on path length rather than time. Consequently, we apply a conditional constraint on the path length that limits the travel distance to a maximum of 235 meters to guarantee quick access to emergency exits. This ensures that more than one exit route is consistently accessible with minimal detours.

Figure 8 highlights the upstream relation between the lecture hall from the previous example as the starting point and all exit discharge doors on the first floor as the controllers. Multiple paths starting from this room lead to an emergency exit door within the specified maximum travel distance. This constraint is evident in the upstream relation, given that nearly all routes are concentrated on the left side of the building. Moreover, these paths further confirm that our algorithm adhered to the specified absolute constraints, and we note that no elevators or restricted areas were encountered along the way. This example illustrates the significance of imposing constraints in spatial network analysis, as it helps narrow the scope of the problem and make it more visually manageable.

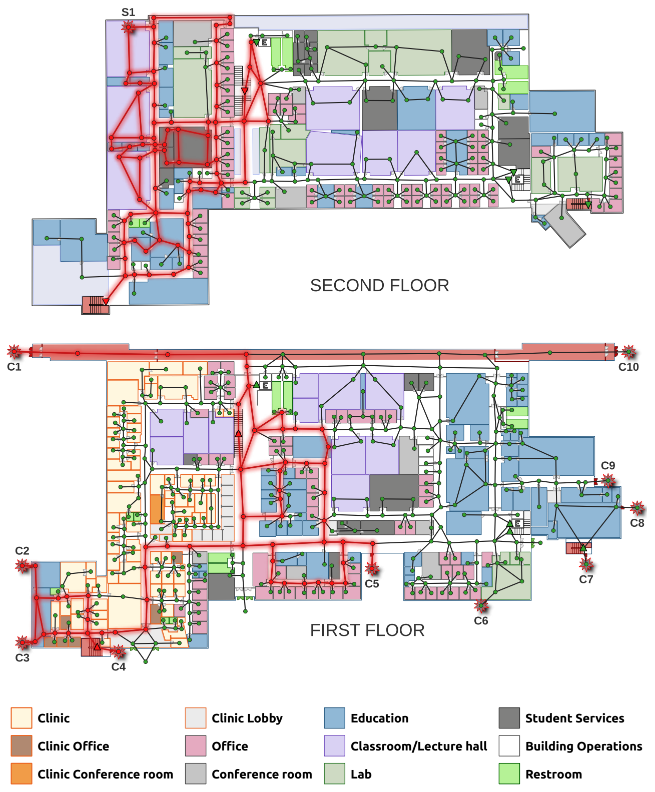


Figure 8: An upstream relation under constraints from a lecture hall of a university campus as the starting point to all emergency exits. Elevators are prohibited, and certain areas require special access, translating to absolute constraints. Additionally, a conditional constraint on the path length limiting the travel distance to a maximum of 235 meters is applied to ensure the emergency exits are reached quickly.

8 Conclusion

With this paper, we have demonstrated how a well-chosen graph search algorithm in a parallel implementation can be used to solve the upstream relation in spatial networks under constraints. This is achieved by integrating three aspects: First, we derive a fast decision algorithm in terms of shortest path queries for the same graph with possibly increasing graph weights; second, we parallelize a linear search through all edges and store the results in a thread-shared array with atomic operations allowing us to apply updates without thread synchronization mechanisms, and, third, we carefully implement the needed resolution of special cases when generalizing from upstream edges to upstream features. Additionally, we provided insights into scaling this approach to handle significantly large networks by minimizing overlap and configuring the thread-local search engines with a few different landmarks to make these threads experts for certain sets of targets.

While our solution is less optimized towards runtime performance, it effectively addresses the challenge of context-sensitive spatial networks. In contrast to related work based on biconnected components [11, 20, 28], our proposed solution is directly capable of including any form of path constraint that essentially translates to increasing edge weights. This is possible with the base algorithm, which allows us to perform constrained shortest path searches. In principle, it is even compatible with notions of upstream, in which a simple path is replaced by an alternative definition of a sensible path. This is provided as long as an efficient decision algorithm for the existence of such a path is available, and this definition is consistent with dividing the path into two segments.

For future work, we envision extending this algorithm to distributed memory parallelism on supercomputers to be able to analyze very large networks beyond the capacity of typical shared memory architectures. In addition, we suggest investigating other types of dynamic constraints, such as edge lifetime contraction. This constraint explores the duration an edge remains accessible under edge contractions until the desired path becomes impassable, and an alternative path must be identified. Understanding how these constraints impact network connectivity could help discover different vulnerabilities and weaknesses of mobility networks under varying conditions in real-world scenarios. This is particularly relevant to implementing robust and adaptive rerouting strategies in the context of mobility networks, where infrastructures like bridges or tunnels may impose constraints such as height or weight limits.

Acknowledgments

We thank the 2018 ACM SIGSPATIAL GIS Cup challenge organizers for formulating this inspiring problem and hosting the challenge.

References

[1] Julhas Alam. 2024. *Bangladeshi leader says a shopping mall that caught fire had no emergency exits. Death toll climbs.* Retrieved May 16, 2024 from https://apnews.com/article/bangladesh-fire-dhaka-mall-32b7fee567511126776d074ddd2d614c

[2] Marc Barthélemy. 2011. Spatial networks. *Physics Reports* 499, 1 (2011), 1–101. https://doi.org/10.1016/j.physrep.2010.11.002

[3] Adil Bhat. 2024. *India's deadly industrial workplaces in the spotlight.* Retrieved May 16, 2024 from https://www.dw.com/en/deadly-industrial-accidents-in-india-kill-and-disable-thousands/a-67930230

[4] Andrzej Białecki, Natalia Jakubowska, Paweł Dobrowolski, Piotr Białecki, Leszek Krupiński, Andrzej Szczap, Robert Białecki, and Jan Gajewski. 2023. SC2EGSet: StarCraft II Esport Replay and Game-state Dataset. *Scientific Data* 10, 1 (Sept. 2023). https://doi.org/10.1038/s41597-023-02510-7

[5] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.

[6] James A. Edwards and Uzi Vishkin. 2012. Better speedups using simpler parallel programming for graph connectivity and biconnectivity. In *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores* (New Orleans, Louisiana) *(PMAM '12)*. Association for Computing Machinery, New York, NY, USA, 103–114. https://doi.org/10.1145/2141702.2141714

[7] Esri. 2023. *Meridian Indoors First Floor.* Retrieved June 04, 2024 from https://www.arcgis.com/apps/mapviewer/index.html?layers=11b9efa3d5fb4337acdbec4e90bbec65

[8] Esri. 2023. *Meridian Indoors Second Floor.* Retrieved June 04, 2024 from https://www.arcgis.com/apps/mapviewer/index.html?layers=968b61bb3d2f4df48dea262bd90b0a49

[9] Robert W. Floyd. 1962. Algorithm 97: Shortest path. *Commun. ACM* 5, 6 (jun 1962), 345. https://doi.org/10.1145/367766.368168

[10] Andrew Goldberg and Chris Harrelson. 2003. Computing the shortest path: A* search meets graph theory. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms* (04 2003). https://doi.org/10.1145/1070432.1070455

[11] Zach Goldthorpe, Jason Cannon, Jesse Farebrother, Zachary Friggstad, and Mario A. Nascimento. 2018. Using biconnected components for efficient identification of upstream features in large spatial networks (GIS cup). In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (Seattle, Washington) *(SIGSPATIAL '18)*. Association for Computing Machinery, New York, NY, USA, 630–633. https://doi.org/10.1145/3274895.3276476

[12] Ruirong Guo, Chaokui Li, and Haibin Guo. 2023. Indoor Navigation Network Model Construction Method Based on Building Information Model. *Journal of Geographic Information System* 15 (01 2023), 367–378. https://doi.org/10.4236/jgis.2023.154018

[13] John Hammock. 2023. The Crucial Role of Infrastructure in Economic Development. https://www.linkedin.com/pulse/crucial-role-infrastructure-economic-development-john-hammock/ Accessed on March 21, 2024.

[14] Peter Hart, Nils Nilsson, and Bertram Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107. https://doi.org/10.1109/tssc.1968.300136

[15] Erik Hoel, Petko Bakalov, Sangho Kim, and Thomas Brown. 2015. Moving beyond transportation: utility network management. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '15)*. Association for Computing Machinery, New York, NY, USA, Article 8. https://doi.org/10.1145/2820783.2820879

[16] Fei Hu, Zhenlong Li, Chaowei Yang, and Yongyao Jiang. 2019. A graph-based approach to detecting tourist movement patterns using social media data. *Cartography and Geographic Information Science* 46, 4 (2019), 368–382. https://doi.org/10.1080/15230406.2018.1496036

[17] Kelsey Jack. 2022. How much do we know about the development impacts of energy infrastructure? https://blogs.worldbank.org/en/energy/how-much-do-we-know-about-development-impacts-energy-infrastructure Accessed on March 21, 2024.

[18] Ben Knight. 2019. *How stable is Germany's power grid?* Retrieved April 26, 2024 from https://www.dw.com/en/berlin-blackout-raises-questions-over-germanys-power-grid/a-47730394

[19] Zhihang Liu, Chenyu Fang, Hao Li, Jinlin Wu, Lin Zhou, and Martin Werner. 2023. Efficiency and equality of the multimodal travel between public transit and bike-sharing accounting for multiscale. *Sustainable Cities and Society* 101 (12 2023), 105096. https://doi.org/10.1016/j.scs.2023.105096

[20] Salles Viana Gomes Magalhães, W. Randolph Franklin, and Ricardo dos Santos Ferreira. 2018. Fast analysis of upstream features on spatial networks (GIS cup). In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (Seattle, Washington) *(SIGSPATIAL '18)*. Association for Computing Machinery, New York, NY, USA, 622–625. https://doi.org/10.1145/3274895.3276474

[21] Dev Oliver, Bo Xu, and Yuanyuan Pao. 2019. ACM SIGSPATIAL cup 2018 - Identifying upstream features in large spatial networks. *ACM SIGSPATIAL Special* 11 (2019), 32–35. https://api.semanticscholar.org/CorpusID:199454201

[22] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. https://networkrepository.com

[23] Tanuja Shanmukhappa, Ivan Wang-Hei Ho, and Chi Tse. 2018. Spatial analysis of bus transport networks using network theory. *Physica A: Statistical Mechanics and its Applications* 502 (02 2018). https://doi.org/10.1016/j.physa.2018.02.111

[24] Yiquan Song, Lei Niu, Pengfei Liu, and Yi Li. 2021. Fire hazard assessment with indoor spaces for evacuation route selection in building fire scenarios. *Indoor and Built Environment* 31 (03 2021). https://doi.org/10.1177/1420326X21997547

[25] Aaron Spray. 2024. *Over 300 Passengers Were Accused Of Bypassing Security Checkpoints In 2023.* Retrieved May 16, 2024 from https://simpleflying.com/passengers-accused-bypassing-security-checkpoints-2023/

[26] Robert Tarjan and Uzi Vishkin. 1985. An Efficient Parallel Biconnectivity Algorithm. *SIAM J. Comput.* 14 (11 1985), 862–874. https://doi.org/10.1137/0214061

[27] Robert Endre Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* 1 (1972), 146–160. https://api.semanticscholar.org/CorpusID:16467262

[28] Thomas C. van Dijk, Tobias Greiner, Bas den Heijer, Nadja Henning, Felix Klesen, and Andre Löffler. 2018. Wüpstream: efficient enumeration of upstream features (GIS cup). In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (Seattle, Washington) *(SIGSPATIAL '18)*. Association for Computing Machinery, New York, NY, USA, 626–629. https://doi.org/10.1145/3274895.3276475

[29] Cola Wang. 2023. *The Importance of Access Control in Airport Security.* Retrieved May 16, 2024 from https://www.aviationpros.com/aviation-security/access-control/article/53059196/jwm-hi-tech-development-co-ltd-the-importance-of-access-control-in-airport-security

[30] Agnieszka Widuto. 2023. EU energy infrastructure: Boosting energy security. https://www.europarl.europa.eu/RegData/etudes/BRIE/2023/753956/EPRS_BRI(2023)753956_EN.pdf Accessed on March 21, 2024.

[31] Liping Yang and Michael Worboys. 2015. Generation of navigation graphs for indoor space. *International Journal of Geographical Information Science* (05 2015). https://doi.org/10.1080/13658816.2015.1041141

[32] Yan Zhou, Yuling Pang, Fen Chen, and Yeting Zhang. 2020. Three-Dimensional Indoor Fire Evacuation Routing. *ISPRS International Journal of Geo-Information* 9, 10 (2020). https://doi.org/10.3390/ijgi9100558