BeRGeR

Using Braids for Byzantine-Resistant Geometric Routing on Polyhedral Networks

Zaz Brown

January 4, 2024

Contents

Problem Definition

Online routing	Nodes are **myopic** (only see immediate neighbors)

Robust routing on low-resource devices

- IoT
- Sensor networks
- Vehicular networks

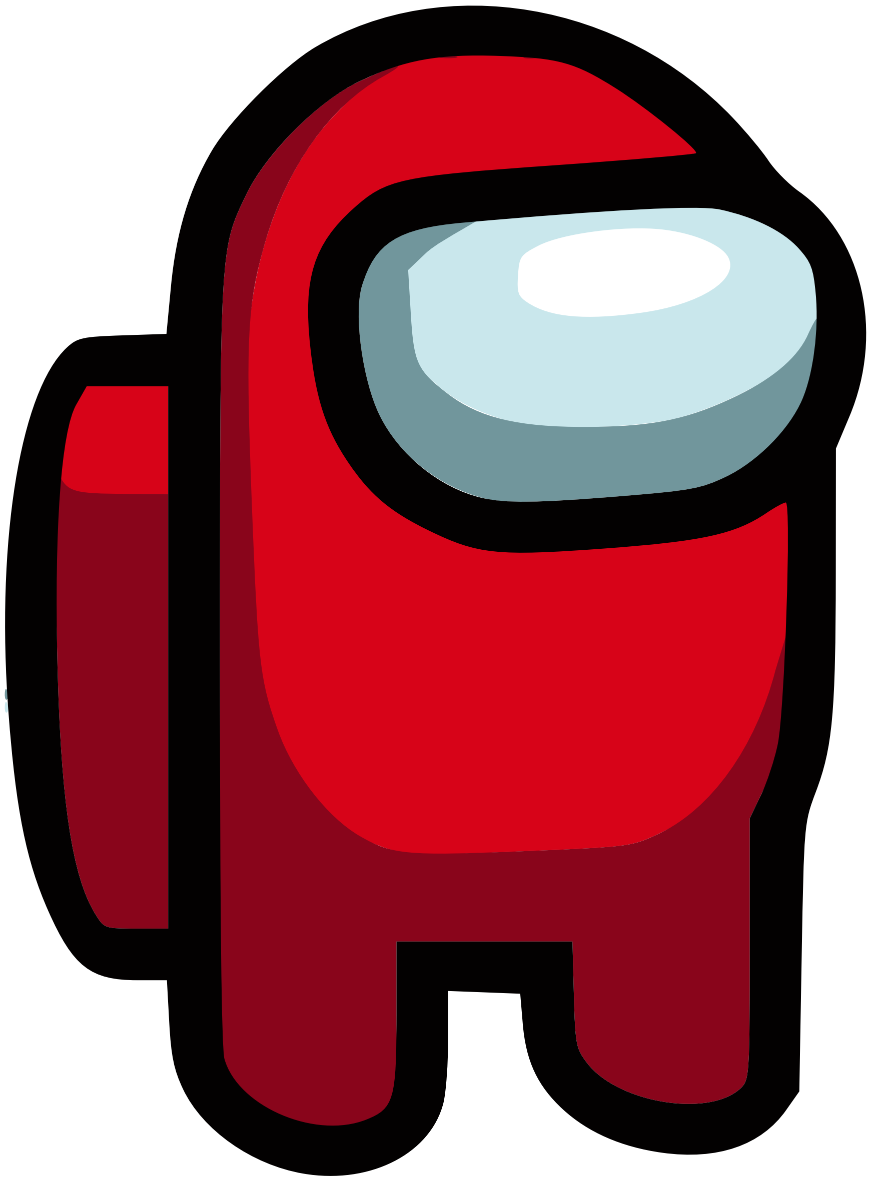
1 faulty "Byzantine" node

Using Braids for Byzantine-Resistant Geometric Routing on Polyhedral Networks

Offline routing	Routing tables (nodes store directions)
Geometric routing	
Greedy routing	Go to node closest to destination
Face routing	Always turn right (or left)

Using Braids for Byzantine-Resistant Geometric Routing on Polyhedral Networks

Byzantine node	Node that behaves arbitrarily



Using Braids for Byzantine-Resistant Geometric Routing on Polyhedral Networks

Network	Graph
Polyhedral	
Planar	No edges intersect
3-Connected	To disconnect the network, you need to remove 3 nodes
3-Connected	$\exists$ 3 disjoint paths between each pair of nodes

Planar $\because$ $\exists$ algorithms to planarize; simplicity **3-connected** $\because$ it is necessary **Menger's theorem**: 3-connected equivalence

Using Braids for Byzantine-Resistant Geometric Routing on Polyhedral Networks

Exactly what they sound like.

Naïve approach

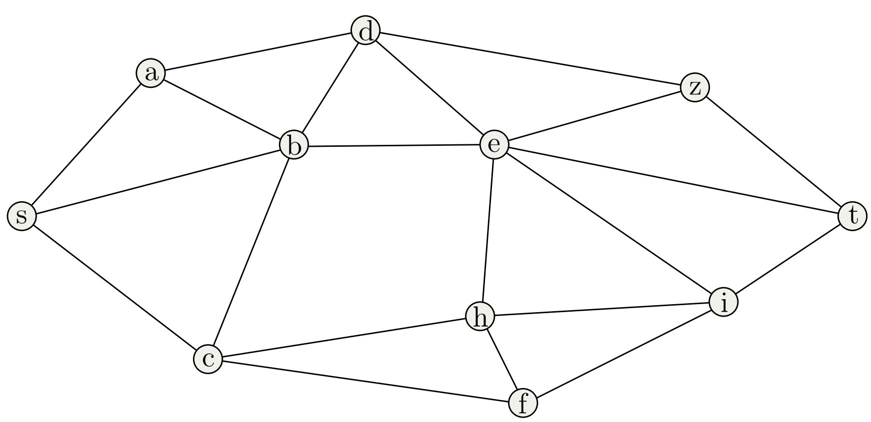
Route along 3 disjoint paths

$$\begin{aligned} \exists i, j \ : \ m_i &= m_j, \\ p_i \ \cap \ p_j &= \emptyset \end{aligned}$$

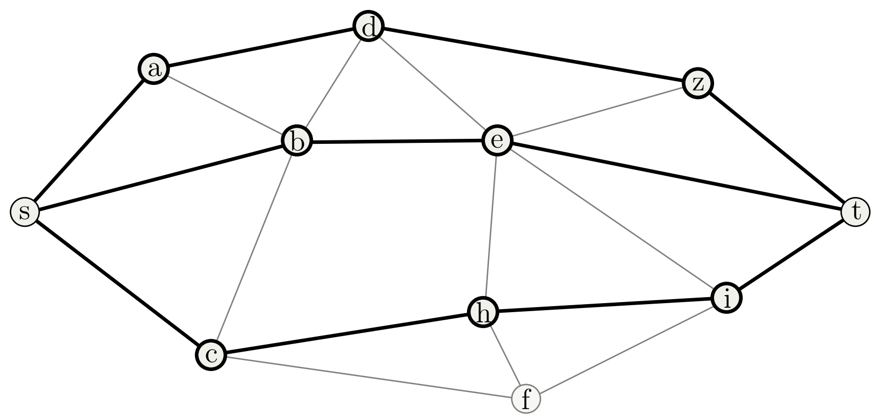
For each node, find 2 disjoint paths that skip it.

We **need** to use 3-connectivity.

Naïve approach



Naïve approach



Hard to do online.

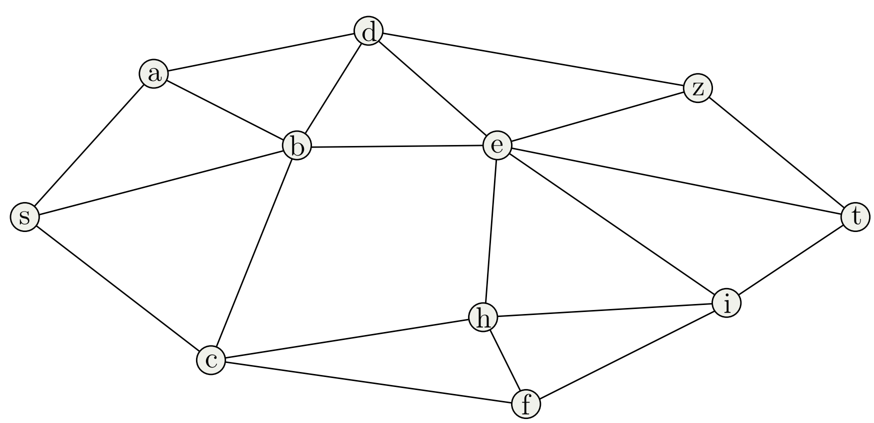
Generalized approach

Route along *collectively* disjoint paths

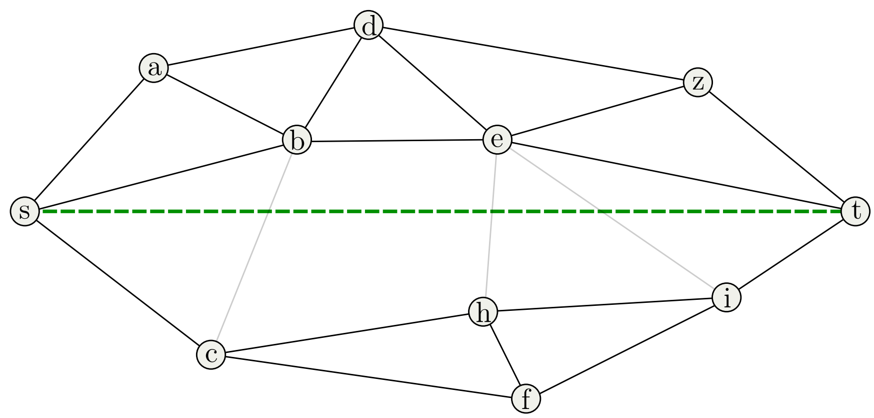
$$\begin{aligned} \exists i, j, k, ... \ :\ & m_i = m_j = m_k = \cdots, \\ & p_i \ \cap \ p_j \ \cap \ p_k \ \cap \ \cdots = \emptyset \end{aligned}$$

For each node, find a set of collectively disjoint paths that skip it.

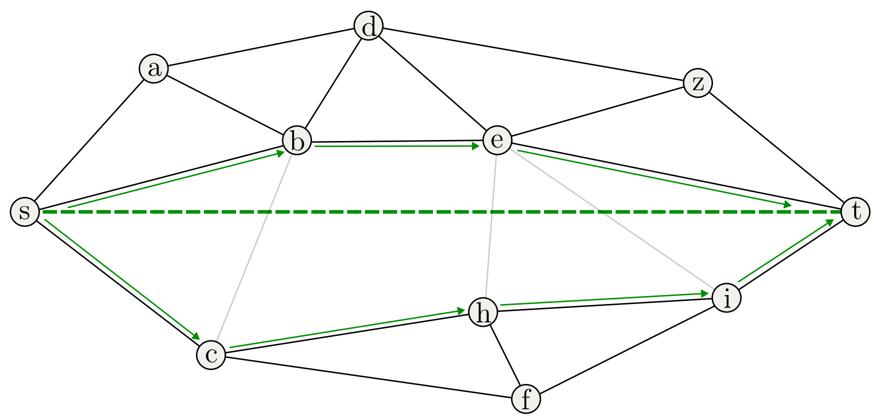
Remove edges that intersect st-line



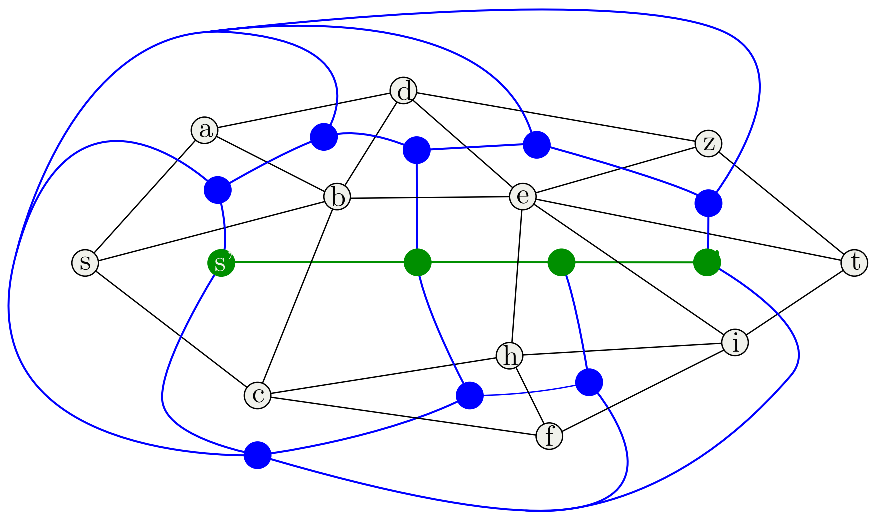
Remove edges that intersect st-line



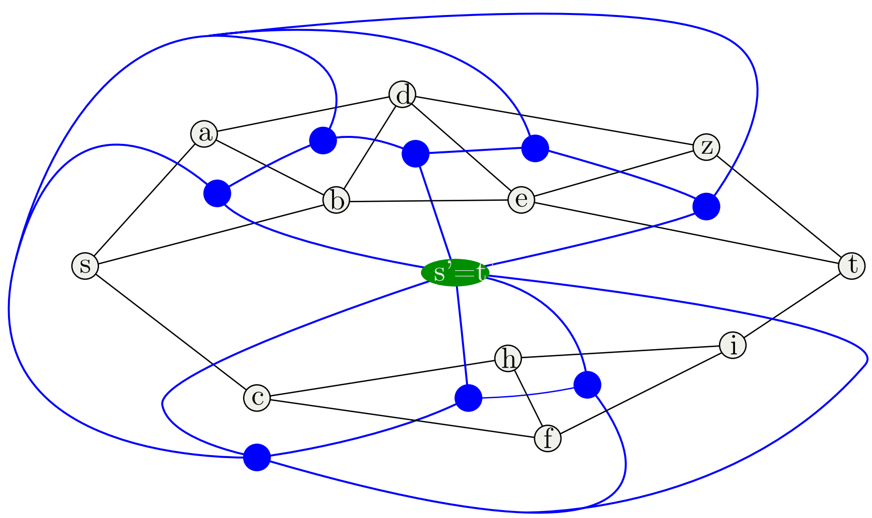
Route along both sides



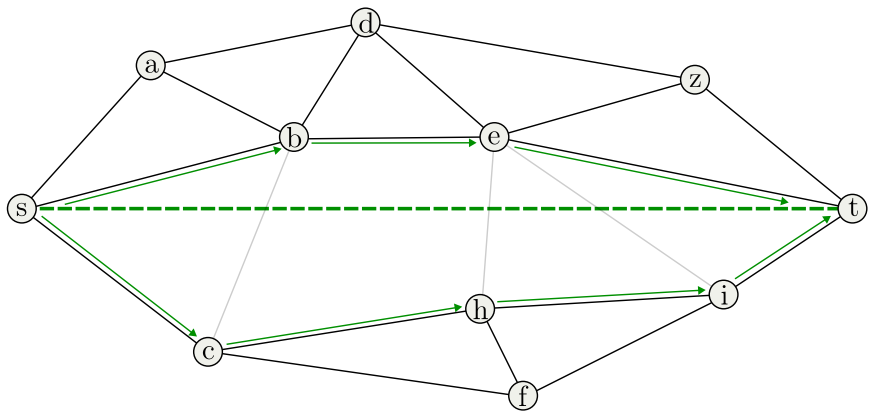
Proof



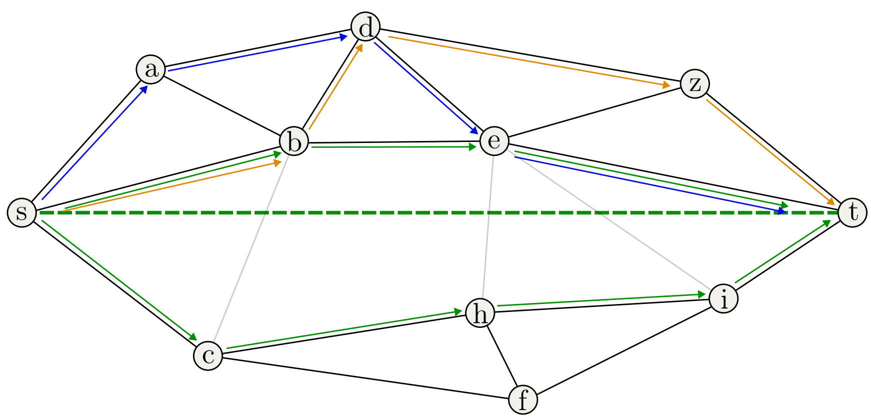
Proof



Route along both sides



Braids



Paper

Reliable Geometric Routing

1 Introduction

We have a network (graph), G, of nodes that know only the current packet, their position, and the set of their neighbors' positions. They do not have space for routing tables, nor space to store information about packets that pass through them. The goal is to route a message from s to t such that t will decode the correct message even in the presence of 1 node that behaves arbitrarily (i.e. is Byzantine).

The simplest solution without a Byzantine node would be to greedy route; at each step going to the node geometrically closest to t. However, local minima result in this naïve algorithm not reaching t. A guaranteed solution is to include draw a line through s and t (encoded in the packet), and then route along all edges of any face that intersects the $s-t$ line. This was done in Concurrent Face Routing (CFR) [1] and is adapted here in Piths / Half-Face Routing as "Half-Face Routing" (HFR) to generate 2 disjoint paths "piths" that form the core of our braided routing algorithm.

The state of the art in geographic routing is GOAFR+ [3] where they route packets within an ellipse, increasing the size of the ellipse if progress is not made. They demonstrate practical efficiency and prove their algorithm is asymptotically optimal in the worst-case: $O(p^2)$, where p is the shortest path length from s to t, and we assume a unit-disk graph (UDG). We attempted something similar, but were stymied by networks that had multiple external faces on the $s-t$ line; we may attempt this again assuming no such intersections.

Finally, [2] proves a correspondence between interactive consensus conditions (ICCs) and error-correcting codes (ECCs). But I believe the ICCs apply to broadcast networks (complete graphs), so are not directly applicable here.

2 Mathematical Formalism

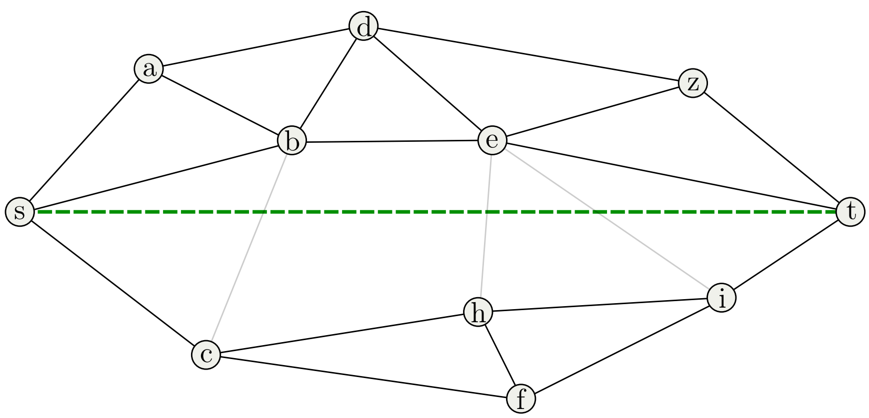
Formally, each node, except for 1, uses the same routing algorithm, f, that takes a message, $m \in M$, the proximal source of the packet, $\mathbf{v}_{-1}$, the current node's position $\mathbf{v}_0$, and the set of the current node's neighbors' positions $\mathbf{V}$ and returns a set of (possibly modified) packets along with the node to pass each packet to:

$$f : m, \mathbf{v}_{-1}, \mathbf{v}_0, \mathbf{V} \mapsto \{(m \in M, \mathbf{v}) : \forall m, \mathbf{v} \in \mathbf{V}\}$$

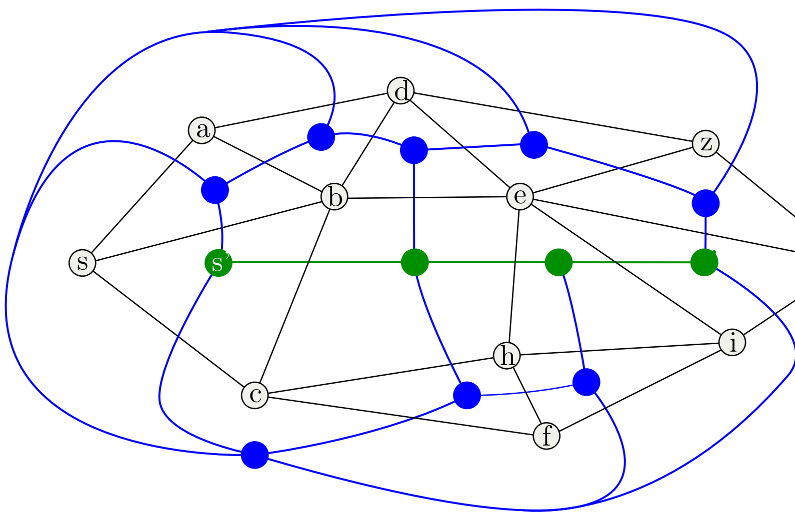
Algorithm 1: BeRGeR: Byzantine Resistant Geographic Routing

```
constants
n                                                        // node coordinates
N                     // circular list of neighbor coordinates sorted clockwise
variables
S                                               // set of packets sent by source
T                                            // set of packets received by target
source node
input:
m                                                                  // message
t                                                                   // target
initial action:
    S.append(t, m)                                      // record the packet at s
    send (m, s, t,  CW, ⊥, ⟨ ⟩) to nextNode (N, t, s, t,  CW, ⊥)   // send CW core
    send (m, s, t, CCW, ⊥, ⟨ ⟩) to nextNode (N, t, s, t, CCW, ⊥)  // send CCW core
                              k := nextNode (N, t, s, t,  CW, ⊥)
    send (m, s, t,  CW,  k, ⟨ ⟩) to nextNode (N, t, s, t,  CW,  k) // send CW braid
                              k := nextNode (N, t, s, t, CCW, ⊥)
    send (m, s, t, CCW,  k, ⟨ ⟩) to nextNode (N, t, s, t, CCW,  k) // send CCW braid
all nodes
receive (m, s, t, c, k, ℓ) from p ⟶
    if k = ⊥ then                                           // if this is a core
        ℓ.append(p)                  // append previous node to list of visited nodes
    if n ≠ t then                                       // if packet is not at target
        if k ≠ p                                 // if sender is not trying to skip itself
         and n ≠ s  or (t, m) ∈ S then              // and packet is not at s and forged
            send (m, s, t, c, k, ℓ) to nextNode (N, p, s, t, c, k)       // forward packet
            if k = ⊥ then                                // if received packet is core
                k := nextNode (N, p, s, t, c, ⊥)                       // next green node
                if k ≠ ⊥ and k ∉ ℓ then                             // if k is unvisited
                    send (m, s, t, c, k, ⟨ ⟩) to nextNode (N, p, s, t, c, k)  // generate braid
    else                                                     // packet reached target
        // core nodes neighboring t
        coreCW ≡ nextNode(N, s, s, t, CCW, ⊥)
        coreCCW ≡ nextNode(N, s, s, t, CW, ⊥)
        // nodes neighboring t next to core nodes (may be the same)
        braidCW ≡ nextNode(N, s, s, t, CCW,  coreCW)
        braidCCW ≡ nextNode(N, s, s, t, CW, coreCCW)
        if p = coreCW and k = ⊥ and c = CW then
            T.append(m, s, CW, ⊥, ℓ)                                  // record CW core
        else if p ∈ {coreCW, braidCW} and c = CW then
            T.append(m, s, CW, k, ⟨ ⟩)                               // record CW braid
        else if p = coreCCW and k = ⊥ and c = CCW then
            T.append(m, s, CCW, ⊥, ℓ)                                // record CCW core
        else if p ∈ {coreCCW, braidCCW} and c = CCW then
            T.append(m, s, CCW, k, ⟨ ⟩)                             // record CCW braid
        if ∃m, s, ℓ1, ℓ2 : {(m, s, CW, ⊥, ℓ1), (m, s, CCW, ⊥, ℓ2)} ⊂ T then
            deliver m                                        // received matching cores
        else if ∃(m, s, c, ⊥, ℓ) ∈ T : ∀i ∈ ℓ, (m, s, c, i, ⟨ ⟩) ∈ T then
            deliver m           // received braids that skip each visited node of a core
```

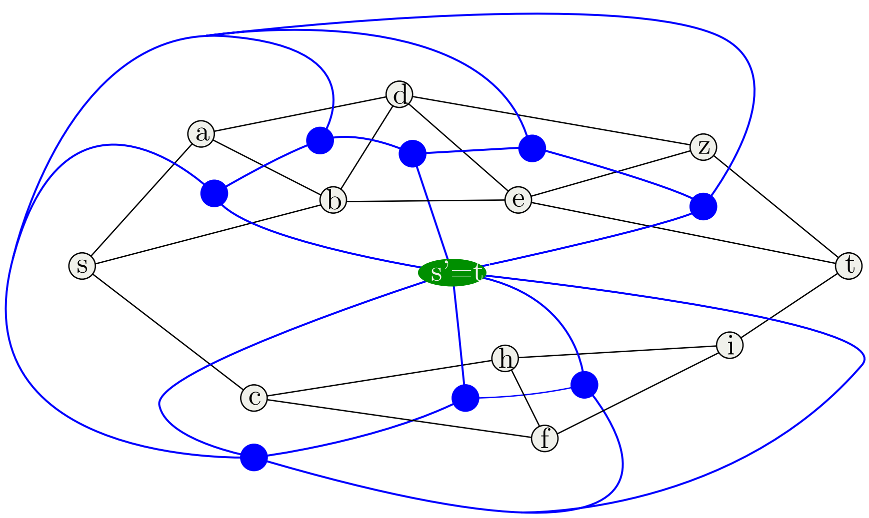
Remove edges that intersect st-line



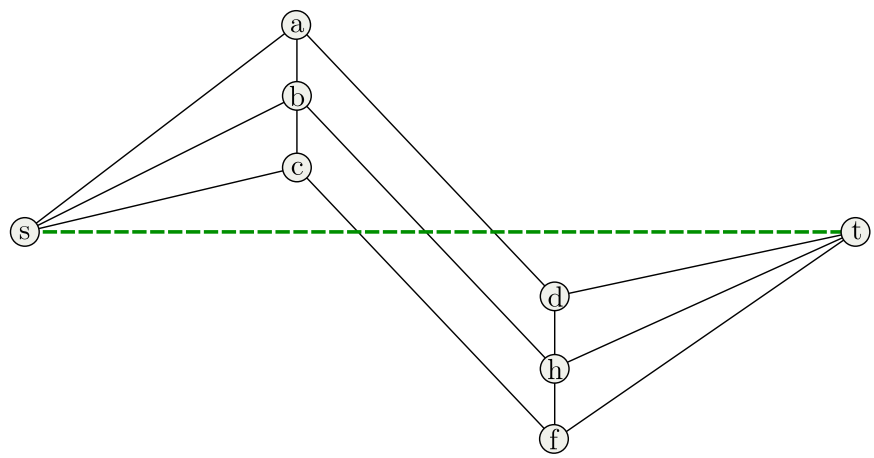
Proof



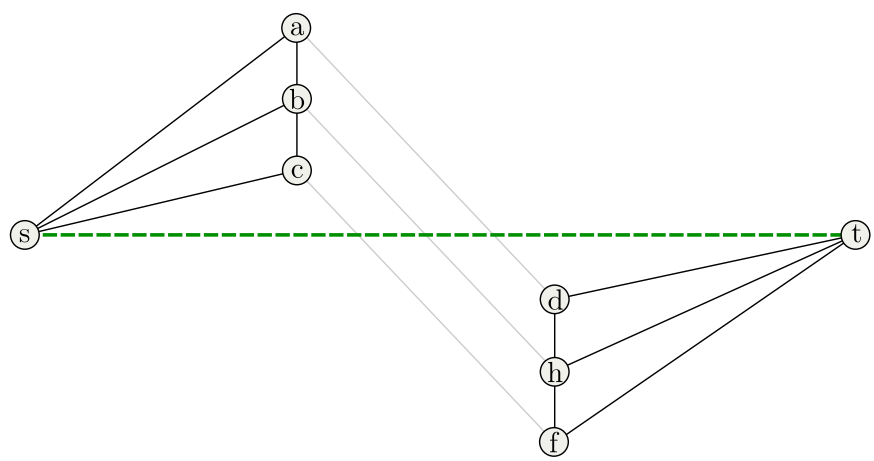
Proof



Counterexample



Counterexample



Acknowledgements

Prof. Mikhail Nesterenko, for supervising the research
Prof. Gokarna Sharma, for his feedback
Prof. Jenya Soprunova, for her feedback
Prof. Darci Kracht, for advice on the presentation

Questions?

zaz@zazbrown.com