TR-RSFB-06-078

Improved Bounds for Online Multi-Path Routing in Faulty Mesh Networks*

Stefan Rührup†
Heinz Nixdorf Institute
University of Paderborn, Germany
sr@uni-paderborn.de

Christian Schindelhauer
Computer Networks and Telematics
University of Freiburg, Germany
schindel@informatik.uni-freiburg.de

April 2006

Abstract

We consider the problem of route discovery in a mesh network with faulty nodes. The number and the positions of the faulty nodes are unknown. It is known that a flooding strategy like expanding ring search can route a message linear in the minimum number of steps d while it causes a traffic (i.e. the total number of messages) of $\mathcal{O}(d^2)$. For optimizing traffic a single-path strategy is optimal producing traffic $\mathcal{O}(d+p)$, where p is the number of nodes that are adjacent to faulty nodes. In TR-RSFB-05-077 "Online Multi-Path Routing in a Maze" we presented a deterministic multi-path online routing algorithm that delivers a message within $\mathcal{O}(d)$ steps causing traffic $\mathcal{O}((d+p)\log^3 h)$. Here, we show an improvement of the traffic bound to $\mathcal{O}((d+p)\log^2 d)$. This algorithm is asymptotically as fast as flooding and nearly traffic-optimal up to a polylogarithmic factor.

*Supported by the DFG Sonderforschungsbereich 376: "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen." and by the EU within the 6th Framework Programme under contract 001907 "Dynamically Evolving, Large Scale Information Systems" (DELIS)

†DFG Graduiertenkolleg 776 (postgraduate programme) "Automatic Configuration in Open Systems"

1 Introduction and Overview

Sending a message is the most fundamental feature of communication networks. We consider two-dimensional mesh networks, which can be found in parallel computers, in integrated circuits, FPGAs (Field Programmable Gate Arrays) and also some kinds of wireless sensor networks. In all these networks nodes may fail or may be unavailable. A node's failure can only be noticed by its neighbors. A straight-forward approach is to regularly test the neighbors of each node, to collect this data and to distribute a map of all failed and working nodes throughout the network. We investigate scenarios where this knowledge is not available when the message is on its way. Due to the lack of global information this routing problem states an online problem.

The basic problem is that the faulty nodes are barriers to the routing algorithm and that the algorithm does not know these barriers. There is no restriction on the size and the shape of the barriers, so even labyrinths are possible. In such situation a fast message delivery can only be guaranteed if every node forwards the message to all neighbors such that the complete network is flooded. This results in a tremendous increase of traffic, i.e. the number of node-to-node transmissions. If the algorithm uses a single-path strategy, then the additional effort necessary for searching a path to the destination, increases the time.

We analyze algorithms with respect to the length of the shortest path d between source and target and with respect to the number of border nodes p, which are the nodes adjacent to faulty nodes. Regarding the time, every single-path online algorithm cannot beat the optimal offline algorithm and in worst case scenarios it has to investigate all the barriers, i.e. a traffic proportional to the number of border nodes p is inevitable. There are single-path algorithms that use only $\mathcal{O}(d+p)$ messages in total, but they need $\mathcal{O}(d+p)$ time steps. Time-optimal algorithms are parallel multi-path algorithms (e.g. expanding ring search) with time $\mathcal{O}(d)$ and traffic $\mathcal{O}(d^2)$ in the worst case.

We are interested in optimizing time and traffic at the same time. One might expect a trade-off situation between these measures. However, our research shows that there are algorithms that approximate the offline time bound and the optimal online traffic bound by a factor of $\mathcal{O}(\sqrt{d})$ [20] at the same time. The quotient comparing to the offline time bound is called the competitive time ratio, while the quotient comparing to the traffic bound of the optimal online algorithm is called the comparative traffic ratio. Subsequent work showed that both bounds could be improved below any polynomial bound: By a to a term of $d^{\mathcal{O}\left(\sqrt{\frac{\log\log d}{\log d}}\right)}$ [21]. This

We call a bound on both ratios the combined comparative ratio (Def. 5).

Strategy	Time	Traffic	Comb. Comp. Ratio
Exp. Ring Search [9, 18]	$\mathcal{O}(d)$	$\mathcal{O}(d^2)$	$\mathcal{O}(d)$
Lucas' Algorithm [13]	$\mathcal{O}(d+p)$	$\mathcal{O}(d+p)$	$\mathcal{O}(d)$
Alternating Strategy [20]	$\mathcal{O}(d^{3/2})$	$\mathcal{O}(\min\{d^2, d^{3/2}+p\})$	$\mathcal{O}(\sqrt{d})$
Selective Flooding [21]	$d \cdot 2^{\mathcal{O}\left(\sqrt{\frac{\log d}{\log\log d}}\right)}$	$\mathcal{O}(d) + p\, d^{\mathcal{O}\left(\sqrt{\frac{\log\log d}{\log d}}\right)}$	$d^{\mathcal{O}\left(\sqrt{\frac{\log\log d}{\log d}}\right)}$
JITE (this paper)	$\mathcal{O}(d)$	$\mathcal{O}((d+p)\log^2 d)$	$\mathcal{O}(\log^2 d)$
Online Lower Bound (cf. [3])	$\Omega(d)$	$\Omega(d+p)$	$\Omega(1)$

With a combined comparative radio of $\mathcal{O}(\log^2 d)$, the JITE algorithm is a break-through in this line of research. More specifically, it delivers a message on a multi-path route within time $\mathcal{O}(d)$ and with traffic $\mathcal{O}(d + p\log^2 d)$. This shows, that one can route a message asymptotically as fast as flooding while increasing the traffic only by a polylogarithmic factor compared to the traffic-optimal online algorithm.

This paper is organized as follows. We continue this section by presenting related research. In the following section we describe the basic definitions and techniques more formally. In Section 3, we present the algorithm starting with an overview and the description of its components. In Section 4, we analyze the time and traffic performance of this algorithm which concludes the paper.

1.1 Related Work

The problem studied in this paper has a strong relation to online search and navigation problems. These problems have been investigated in different research communities, which Angluin et al. [1] called "the online competitive analysis community" and the "theoretical robotics community". The fundamental goal in online searching is to find a point in an unknown environment. In theoretical robotics the scenarios contain obstacles of arbitrary shape and the performance of algorithms is expressed by comparing the distance traveled by the robot to the sum of the perimeters of the obstacles [15, 1] (see also [2] for a survey and [14, 19] for an overview of path-planning and maze traversal algorithms). The competitive analysis community has studied various kinds of scenarios with restrictions on the obstacles (e.g. quadratic, rectangular or convex obstacles). The performance is expressed by the *competitive ratio*, which is the ratio of the distance traveled by the robot and the length of the shortest obstacle-free path to the target [17, 3].

Our model connects these two lines of research. The connection between the scenarios considered in online searching and our model is obvious: Scenarios with a lower bound on distance between start point and target point and with finite obstacle perimeters can be modeled by a faulty mesh network. We also investigate the problem of searching a point in an unknown environment, but here, the search can also be done in parallel. For robot navigation problems it is not clear how unbounded parallelism can be modeled in a reasonable way. Usually, navigation strategies are only considered for a constant number of robots. Therefore, we consider a mesh network with faulty parts as underlying model, which enables us to study the impact of parallelism on the time needed for finding the target. For the time analysis we use the *competitive ratio* as used by the competitive analysis community. On the other hand the traffic is compared to the perimeters of the barriers which gives the *comparative traffic ratio*. This ratio expresses the amount of parallelism used by the algorithm.

Routing in faulty networks has also been considered as an offline problem. In the field of parallel computing the the fault-tolerance of networks is studied, e.g. by Cole et al. [8]. The problem is to construct a routing scheme in order to emulate the original network. Zakrevski and Karpovski [24] investigate the routing problem for two-dimensional meshes. The model is similar to ours as they consider two-dimensional meshes under the store-and-forward model. Their algorithm needs an offline pre-routing stage, in which fault-free rectangular clusters are identified. Routing algorithms for two-dimensional meshes, that need no pre-routing stage are presented by Wu [22]. These algorithms use only local information, but the faulty regions in the mesh are assumed to be be rectangular blocks. In [23] Wu and Jiang present a distributed algorithm that constructs convex polygons from arbitrary fault regions by excluding nodes from the routing process. This is advantageous in the wormhole routing model, because it helps to reduce the number of virtual channels. We will not deal with virtual channels and deadlock-freedom as we consider the store-and-forward model.

Bose and Morin [6, 5] study the online routing problem for triangulations and plane graphs with certain properties and present constant-competitive algorithms for routing in these graphs. In triangulations, where no local minima exist, routing can be done by a greedy strategy. Such strategies are also used for position-based routing.

Position-based routing is a reactive routing used in wireless networks, where the nodes are equipped with a positioning system, such that a message can be forwarded in the direction of the target (see [16]

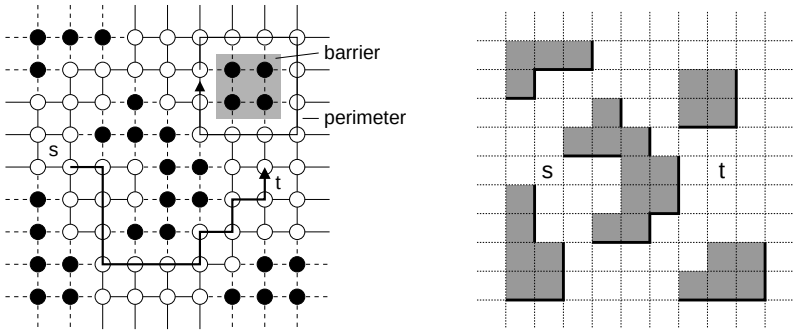

Figure 1: Left: Mesh network with faulty nodes (black), routing path and right-hand traversal path. Right: Abstract representation of the same network.

for a survey). Due to the limited range of the radio transceivers, there are local minima and messages have to be routed around void regions (an analog to the fault regions in the mesh network). There are various single-path strategies, e.g. [11, 7, 12]. Position-based routing strategies have been mainly analyzed in a worst case setting, i.e. the void regions have been constructed such that the connections form a labyrinth. In this case the traffic-efficient single-path strategies produce as much traffic as flooding. In our analysis we take the perimeters of fault regions into account, so we can express performance beyond the worst case point of view.

This paper improves previous results of the authors: In [20] an algorithm is presented that uses a single-path search strategy and flooding alternatingly and has a combined comparative ratio of $\Theta(d^{1/2})$. A reduction of this ratio to $d^{\mathcal{O}\left(\sqrt{\frac{\log\log d}{\log d}}\right)}$ could be achieved by an algorithm that uses a grid subdivision of the search area and floods only the grid squares, if the speed of flooding is needed and justified by many barriers [21].

2 Basic Definitions and Techniques

A two-dimensional mesh network with faulty nodes is defined by a set of nodes $V \subseteq \mathbb{N} \times \mathbb{N}$ and a set of edges $E := \{(v, w) : v, w \in V \wedge |v_x - w_x| + |v_y - w_y| = 1\}$. A node v is identified by its position $(v_x, v_y) \in \mathbb{N} \times \mathbb{N}$ in the mesh. There is no restriction on the size of the network, because time and traffic are analyzed with respect to the position of the given start and target node in the network. We will see that the major impact on the efficiency of the routing algorithm is not given by the size of the network.

We assume a synchronized communication: Each message transmission to a neighboring node takes one *time step*. For multi-hop communication we assume the messages to be transported in a store-and-forward fashion. We also assume that the nodes do not fail while a message is being transported. Otherwise, a node could take over a message and then break down. However, there is no global knowledge about faulty nodes. Only adjacent nodes can determine whether a node is faulty.

2.1 Barriers, Borders and Traversals

The network contains *active* (functioning) and *faulty* nodes. Faulty nodes neither participate in communication nor can they store information. Faulty nodes which are orthogonally or diagonally neighboring form a *barrier*. A barrier consists only of faulty nodes and is not connected to or overlapping with other barriers. Active nodes adjacent to faulty nodes are called *border nodes*. All the nodes in

the neighborhood (orthogonally or diagonally) of a barrier B form the *perimeter* of B. A path around a barrier in (counter-)clockwise order is called a *right-hand (left-hand) traversal path*, if every border node is visited and only nodes in the perimeter of B are used. The *perimeter size* $p(B)$ of a barrier B is the number of directed edges of the traversal path. The *total perimeter size* is $p := \sum_{i \in \mathbb{N}} p(B_i)$. The perimeter size is the number of steps required to send a message from a border node around the barrier and back to the origin, whereby each border node of the barrier is visited. It reflects the time consumption of finding a detour around the barrier.

2.2 The Competitive Time Ratio

Time is the number of steps needed by the algorithm to deliver a message, which is equivalent to the length of a path a message takes. Comparing the time of the algorithm with the optimal time leads to the competitive ratio. This so-called *competitive analysis* is well known in the field of online algorithms [4].

Definition 1 *An algorithm A has a* competitive ratio *of c, if* $\forall x \in \mathcal{I} : C_A(x) \leq c \cdot C_{\mathrm{opt}}(x)$, *where $\mathcal{I}$ is the set of all instances of the problem, $C_A(x)$ the cost of algorithm A on input x and $C_{\mathrm{opt}}(x)$ the cost of an optimal offline algorithm on the same input.*

We compare the time of the algorithm with the length d of the shortest path to the target. Note, that the shortest path uses only non-faulty nodes.

Definition 2 *Let d be the length of the shortest barrier-free path between source and target. A routing algorithm has* competitive time ratio $\mathcal{R}_t := T/d$ *if the message delivery is performed in T steps.*

2.3 The Comparative Traffic Ratio

Traffic is the number of messages the algorithm produces. Regarding traffic, a comparison with the best offline behavior would be unfair, because this bound cannot be reached by any online algorithm. So, we define a *comparative ratio* based on a *class* of instances of the problem, which is a modification of the comparative ratio introduced by Koutsoupias and Papadimitriou [10] (see the Definition in [21]):

Definition 3 *An algorithm A is has a* comparative ratio $f(P)$, *if*

$$\forall p_1 \dots p_n \in P : \max_{x \in \mathcal{I}_P} C_A(x) \leq f(P) \cdot \min_{B \in \mathcal{B}} \max_{x \in \mathcal{I}_P} C_B(x),$$

where $\mathcal{I}_P$ is the set of instances which can be described by the parameter set P, $C_A(x)$ the cost of algorithm A and $C_B(x)$ the cost of an algorithm B from the class of online algorithms $\mathcal{B}$.

With this definition we address the difficulty that is caused by a certain class of scenarios that can be described in terms of the two parameters d and p. For any such instance the online traffic bound is $\min_{B \in \mathcal{B}} \max_{x \in \mathcal{I}_{\{d,p\}}} C_B(x) = \Theta(d + p)$. Note, that for any choice of a scenario one can find an optimal offline algorithm: $\max_{x \in \mathcal{I}_{\{d,p\}}} \min_{B \in \mathcal{B}} C_B(x) = d$. This requires the modification of the comparative ratio in [10] in order to obtain a fair measure. So, we use the online lower bound for traffic to define the *comparative traffic ratio*.

Definition 4 *Let d be the length of the shortest barrier-free path between source and target and p the total perimeter size. A routing algorithm has* comparative traffic ratio $\mathcal{R}_{Tr} := M/(d + p)$ *if the algorithm needs altogether M messages.*

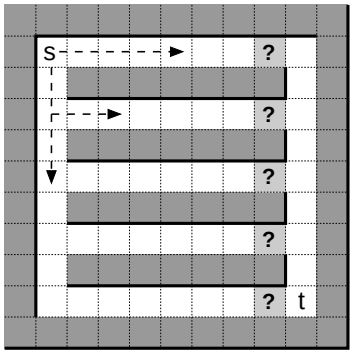

Figure 2: The lower bound scenario of Theorem 1. The adversary controls the "exits" which are marked with "?".

The combined comparative ratio addresses both the time efficiency and the traffic efficiency:

Definition 5 *The* combined comparative ratio *is the maximum of the competitive time ratio and the comparative traffic ratio:* $\mathcal{R}_c := \max\{\mathcal{R}_t, \mathcal{R}_{Tr}\}$

2.4 Lower Bounds

An offline algorithm, which has global knowledge, can determine a path to the target in time $\mathcal{O}(d)$ and with $\mathcal{O}(d)$ messages, because it does not need to examine multiple paths. Online algorithms have to search for the target and this can be hindered by barriers. The following theorem shows the online lower bound for traffic. A similar proof is given by Blum et al. [3] for the competitive ratio of online path planning algorithms. A lower bound for path planning algorithms with respect to the perimeter size is also stated by Lumelsky [14].

Theorem 1 *Every online routing algorithm needs at least* $\Omega(d + p)$ *messages, where* d *is the length of the shortest barrier-free path and* p *the total perimeter size.*

Proof: We consider a scenario with k barriers of size $1 \times l$. These barriers are aligned in parallel, forming long narrow corridors (see Figure 2). At the end of each corridor an adversary may place a barrier or not (indicated with a question mark in the figure). Only one corridor has to remain open. The algorithm can detect an exit only by standing in front of it, i.e. it has to examine the whole corridor. For the length of the shortest path d holds $l < d \leq l + 2k + \mathcal{O}(1)$. The total perimeter size is given by the the perimeters of all obstacles and the enclosure, therfore $p = k \cdot 2l + \mathcal{O}(1)$. With less than $\frac{p}{2} - \mathcal{O}(1)$ messages only half of the corridors can be examined. If the adversary chooses the exit at random, then with probability $\frac{1}{2}$ every online routing strategy must fail to reach the target with less than $\frac{p}{2} - \mathcal{O}(1)$ messages. ■

As a single-path strategy needs one message for each step, this has the following consequence.

Corollary 1 *Every single-path online routing algorithm needs* $\Omega(d + p)$ *time steps, where* d *is the length of the shortest barrier-free path and* p *the total perimeter size.*

2.5 Basic Strategies

Lucas' Algorithm: A simple single-path strategy is the following *barrier traversal* algorithm (Lucas' algorithm, [13]): (1.) Follow the straight line connecting source and target node. (2.) If a barrier is in the way, then traverse the barrier, remember all points where the straight line is crossed, and resume step 1 at that crossing point that is nearest to the target.

This algorithm needs at most $d + \frac{3}{2}p$ steps, where d is the length of the shortest barrier-free path and p the sum of the perimeter lengths of all barriers. This bound is asymptotically optimal [14] and holds also for the traffic, because this is a *single-path* strategy.

Expanding Ring Search: A straightforward multi-path strategy is *expanding ring search* [9, 18], which is nothing more than to start flooding with a restricted search depth and repeat flooding while doubling the search depth until the destination is reached. This strategy is asymptotically time-optimal, but it causes a traffic of $\mathcal{O}(d^2)$, regardless of the presence of faulty nodes.

Continuous Ring Search: We modify this strategy as follows: The source starts flooding without a depth restriction, but with a delay of σ time steps for each hop. If the target is reached, a notification message is sent back to the source. Then the source starts flooding a second time, and this second wave, which is not slowed down, is sent out to stop the first wave. This *continuous ring search* needs time $\mathcal{O}(d)$ and causes a traffic of $\mathcal{O}(d^2)$, which is no improvement to expanding ring search. But an area is flooded at most two times, whereas expanding ring search visits some areas $\mathcal{O}(\log d)$ times. We use this advantage for our algorithm.

Lemma 1 *Let d be the length of the shortest path connecting s and t. Continuous ring search with a slow-down of $\sigma > 1$ finds a path in time $\sigma \cdot d$ and produces a traffic of at most $\mathcal{O}\left((\frac{\sigma+1}{\sigma-1}\, d)^2\right)$ where d is the length of the shortest path connecting s and t.*

Proof: The first wave is slowed down by a factor of σ and reaches the target in time σd, where d is the lengths of the shortest path. This wave proceeds at a speed of $1/\sigma$ until it is stopped by the second wave. The second wave is started after a time of $\sigma d + d$ when the notification from the target has reached the source. The notification and the second wave travel at a speed of 1. Both waves meet at time step t when $t/\sigma = t - \sigma d - d$. This solves to $t = \frac{\sigma(\sigma+1)}{\sigma-1} d$. At this time the first wave has reached a distance of $t/\sigma = \frac{\sigma+1}{\sigma-1} d$. ■

3 The JITE Algorithm

In this section the JITE Algorithm (Just-In-Time Exploration) is presented. First, an overview is given, followed by a detailed description of the most important part: the Fast Exploration Algorithm.

3.1 Overview

The algorithm starts with a search area consisting of four connected quadratic subnetworks, called *squares*, with s lying on the common corner (see Figure 3). If the target cannot be found inside this search area, new quadratic subnetworks in the environment are investigated. So, the search area is enlarged by a factor of 3 until the target is reached. Therefore we define $\log n := \log_3 n$. We can assume that $||s - t||_\infty = 3^k$ for some $k \in \mathbb{N}$, such that the target lies on the border of a square. If this

is not the case we search for any node s' with $||s' - t||_\infty = 3^k$ with the same algorithm and restart the algorithm from s'. This increases time and traffic only by a constant factor.

For the expansion of the search area we use the idea of continuous ring search: When the target is found, the source is notified, which sends out messages to stop the search. The difference of this algorithm to continuous ring search is that not the whole area is flooded. Instead of flooding, the algorithm uses a modified breadth-first search. This BFS uses only certain paths, which are restricted to the borders of squares, called *frames*, and the nodes adjacent to barriers. These paths are determined by an *exploration* strategy before they are visited by the BFS. If a square contains few barriers, then it can be traversed easily by a single path. Otherwise it has to be subdivided and examined using multiple paths. This way the exploration strategy decides which paths are used by the BFS.

Exploration is done by following the border of a square (frame). If there are barriers intersecting the frame we have to use paths in the interior of the square. A left-hand or right-hand traversal of the barriers may produce a path that is much longer than the shortest path. To keep this overhead small, we try to traverse the interior of a square in a bounded time that depends on the frame size. If this is not possible, then we subdivide the square into 9 smaller squares and try the traversal again. This recursive subdivision yields the paths that can be used by the BFS and provides an approximation of the shortest path.

The exploration process is triggered by the BFS. The leaves of the BFS tree define the border of the examined area. We call this border the *shoreline*. The shoreline starts after a delay, such that there is enough time for the exploration of the initial frames. The shoreline uses only partitions of frames that are already explored and that contain few barriers (simple partitions). When the shoreline enters a frame, it triggers the exploration in neighboring frames. Exploration is only performed in the proximity of the shoreline and squares are explored just-in-time. As the exploration takes some time, we slow down the shoreline by a constant factor, such that the squares in proximity to the shoreline can be explored in time. Therefore we call these two strategies *Slow Search* and *Fast Exploration*. The size of the frames to be explored is proportional to their distance to the shoreline. As the time needed for the exploration is proportional to the size of the square, the exploration can be done in time.

3.2 Fast Exploration

The BFS uses the borders of quadratic subnetworks, called frames, which were examined by the exploration process. The *frame* of a $g \times g$ mesh is the set of framing nodes $F = \{v \in V_{g \times g} : v_x \in \{1, g\} \vee v_y \in \{1, g\}\}$. The exploration process examines a frame by starting a *traversal*[1] of the frame nodes and the border nodes (nodes adjacent to barrier nodes). As a frame can be partitioned by a barrier, we refer to a *partition* of a frame, which is enclosed by frame nodes and barrier nodes (see Figure 4). A *right-hand traversal path* in a partition of a frame is a path containing all frame nodes of the partition and all border nodes (adjacent to barriers), where the nodes are visited in counter-clockwise order. We call a partition of a frame *simple*, if it is not intersected by complicated barriers. Therefore we require that a fraction of the frame nodes (expressed by g/γ) is accessible and that there are not too many border nodes.

Definition 6 *A partition of a $g \times g$ frame is* simple*, if it contains a frame node v and a right-hand traversal path starting at v contains at least $4(g - \frac{g}{\gamma})$ frame nodes and at most g/γ border nodes.*

The factor γ is determined at the time t when the exploration starts. In the analysis this is expressed by the notation $\gamma(t)$.

[1] A traversal uses the well-known *right-hand rule*: By keeping the right hand always in touch of the wall, one will find the way out of the maze.

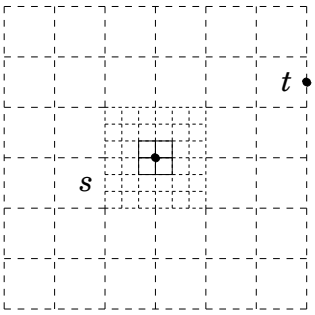

Figure 3: Initial frames (solid) and extended search area.

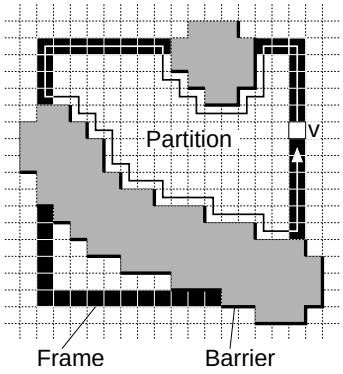

Figure 4: Partition of a frame, defined by a right-hand traversal path.

3.2.1 Subdivisons

Non-simple partitions are recursively subdivided, until there are only simple partitions. A *subdivision* of a frame F is constructed as follows: If F contains a partition that is not simple, then subdivide F into nine equally-sized sub-frames and apply this rule recursively to the sub-frames. A subdivision is called *perfect*, if it contains no frames with side length $g > 3$ that are intersected by a barrier.

For the difference in the size of neighboring frames a restriction is required. So, a subdivision of a frame triggers a subdivision of neighboring frames if necessary. This is expressed by the following rule:

Subdivision Rule: A simple partition of a $3g \times 3g$ frame is subdivided

1. if there is an orthogonally neighboring frame of size $g \times g$
2. if there is diagonally neighboring frame of size $\frac{g}{3} \times \frac{g}{3}$.

The 3×3-subdivision is used instead of a 2×2-subdivision because the latter suffers from a domino effect in the initial situation (see Figure 10): If the search area is enlarged, then the side length for the new squares is doubled. Then a subdivision of a small square near the source, which may be caused by a small barrier, is sufficient to trigger a cascade of subdivisions. This has negative consequences on the traffic.

The Frame Exploration Algorithm

The exploration of a frame is started from one or more frame nodes, which are called *entry nodes*. In each initial frame the entry node is the source node, in other frames the exploration is triggered by the shoreline and then the first nodes that receive a notification message (in the fifth round, see below) become entry nodes.

The exploration of a frame basically consists of a "round-trip communication", i.e. a frame node injects two *traversal messages*, one is sent on a right-hand traversal path, the other on a left-hand traversal path along the frame and the barriers that intersect the frame (a circular tour along the interior of the frame as shown in Figure 4). The ideal case is that these messages meet at some node, but that is not always possible. On the one hand, complicated barriers can prevent further forwarding. However, the criterion for simple partitions (see Def. 6) has to be checked without following long traversal paths along barriers. In a $g \times g$ square without barriers each message travels $2g$ steps along the frame. In

the presence of barriers a detour of at most g/γ steps is allowed, so each message can be stopped after $2(1 + 1/\gamma)g$ steps. The two messages carry a counter for visited frame nodes and border nodes, such that the criterion for a simple partition can be checked when they meet. On the other hand, the exploration of a frame is not always started by a single frame node. So we have to find a mechanism for concurrent exploration processes. Therefore, we need more than one round to collect the information and distribute it to the participating nodes.

1st round, "wake-up": Each entry node generates one message, called *wake-up message* that is sent on a left-hand traversal path around the current partition. A wake-up message stops if it encounters another entry node that already started a message of the same type or if it travels more than $4(1 + 1/\gamma)g$ steps. If an entry node does not receive a wake-up message after $4(1 + 1/\gamma)g$ time steps, then this is a proof for too many barriers. But the reverse is not true. Only if the original message is returned to the originating entry node, then this proves that the current partition is simple. Note, that if the message was stopped by another entry node, it is unclear which case is true. Therefore, the following rounds are needed. One node has to be elected as the *coordinator* for the next round. In this round, the first frame node in counter-clockwise order after the left upper corner that is accessible in the current partition becomes the coordinator for the second round. It is the first frame node on the left-hand traversal path after visiting the last frame node on the left side of the frame. If the left side is obstructed by barriers, then the partition is not simple and then there is no coordinator for the second round Also, if the traversal fails then there is no coordinator. In this case each entry node knows that the partition is not simple after waiting until the third round.

2nd round, "count": The coordinator, which is triggered by a wake-up message in the first round, generates two "counting" messages. These messages travel clockwise and counter-clockwise on the border nodes and carry a counter for visited frame nodes and border nodes. If the messages meet and if they counted at most g/γ border nodes and at least $4(g - \frac{g}{\gamma})$ frame nodes, then the partition is simple and the node where they meet becomes the coordinator for the third round. Otherwise there is no coordinator. So, a simple partition can be checked within $2(1 + 1/\gamma)g$ steps.

3rd round, "stop": If the partition is simple, then the coordinator injects two traversal messages to inform all nodes on the traversal path that the partition is simple and no further subdivisions are necessary. These messages meet at some node, which becomes coordinator for the fourth round. If the partition is not simple, then there is no coordinator and no further messages will be produced. From the lack of the third (or the second) round message all entry nodes of the current partition notice (after waiting $2(1 + 1/\gamma)g$ time steps) that the partition is not simple and has to be subdivided. As the entry nodes know the arrival time of the shoreline, they will wait (if necessary) and begin with the first round of the Frame Exploration Algorithm just in time. To ensure that there is enough time for the exploration of the sub-squares, a pause of Δ time steps after the third round is necessary.

Since the messages used in these three rounds are not available at all frame nodes at the same time, it could happen that another part of the shoreline enters the frame from another side and trigger a subdivision. Other parts of the shoreline and the explorations of smaller squares are not stopped by the messages of these three rounds. We will see that this part of the shoreline does not need to enter a simple partition. The shoreline and any exploration of smaller frames are not stopped by any of these messages. On the other hand, any exploration of smaller frames stops any of these three messages.

4th round, "close": The coordinator sends two traversal messages to close the partition, i.e. that all explorations inside the partition and also parts of the shoreline that have entered the frame from another side are stopped. The processes in the interior can be stopped by simply sending stop messages that follow the paths of the BFS tree from the frame nodes. Actually, this precaution is not necessary as long as the processes in the interior of the partition do not interfere with the outside of the frame. If the shoreline reaches a closed partition, it follows the border of the partition.

After the fourth round the exploration ends. The first round takes $4(1 + 1/\gamma)g$ time steps, each successive round $2(1 + 1/\gamma)g$ time steps plus the additional pause time Δ after the third round. Choosing $\Delta = (1 + 1/\gamma)g$ time steps is sufficient for the following reason: After the third round the entry nodes know that the partition is not simple. So in the remaining $\Delta + 2(1 + 1/\gamma)g$ time steps until the end of round four there must be enough time for performing four rounds of the exploration of a $\frac{g}{3} \times \frac{g}{3}$ sub-square. So, $\Delta + 2(1 + 1/\gamma)g \geq 10(1 + 1/\gamma)\frac{g}{3} + \frac{\Delta}{3} \Rightarrow \Delta \geq 2$. So, the exploration of a simple partition takes $12(1 + \frac{1}{\gamma})$ time steps. The induced traffic amounts to $16(1 + \frac{1}{\gamma})g$.

5th round, "notify": A fifth round begins, when the shoreline enters a simple partition, which is already explored. Then the exploration in the neighboring frames has to be initiated. For this purpose, the node that is reached first by the shoreline sends two traversal messages that trigger the frame nodes of neighboring frames in order to start the exploration. These messages are stopped by other messages of this type.

All frame nodes on the border of unexplored frames will be the new entry nodes. The fifth message gives an estimation when the shoreline will reach this point (at the latest). Let t be the time that this notify message needs to reach the new entry nodes. From the considerations in Lemma 2 follows that this time is bounded by $t \geq \frac{g}{3} - \frac{g}{\gamma} = g\frac{\gamma-3}{3\gamma}$. So the frame size g' of the neighboring frame is chosen such that g' is the largest power of 3 such that $g' \leq \frac{3\gamma t}{\gamma-3}$.

We will choose γ depending on the length of the shortest path. Since this information is not known beforehand we have to use an approximation which is given by the progress of the shoreline. So we choose γ depending on the elapsed time t and denote this by $\gamma(t)$ in the following.

4 Analysis

In the following, time and traffic of the JITE Algorithm are analyzed separately. The time bound can be achieved because Fast Exploration constructs a path network that approximates the shortest path. This construction is done just in time. Slow Search uses this path system and is delayed only by a constant factor. The traffic bound follows from the message paths that form the recursive subdivision of the search area.

4.1 Time

For the time behavior we measure the time that the shoreline (BFS) needs to reach the target. We can rely on the fact that the shoreline proceeds on already explored frame nodes, since the "notify"-messages trigger entry nodes to explore frames. If a frame is subdivided by the exploration process, then neighboring frames that are not already explored are subdivided according to the Subdivision Rule (see Section 3.2.1). The following observation follows directly from this rule:

Observation 1 *A simple partition of a $3g \times 3g$ frame is* not *subdivided into smaller frames*

1. *if all orthogonally neighboring frames are never subdivided below a size of $g \times g$ and contain only simple partitions and*
2. *if all diagonally neighboring frames are never subdivided below a size of $\frac{g}{3} \times \frac{g}{3}$ and contain only simple partitions.*

If the search area is expanded, the size of the new squares is increased by a factor of at most 3. So, there are no subdivisions of these new squares only because of the Subdivision Rule, unless they are

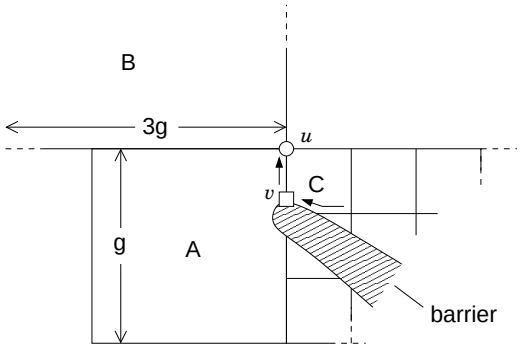

Figure 5: The shoreline enters square A at node v. The exploration of square B must be finished, when the shoreline enters B.

intersected by barriers. The restriction on the size of neighboring squares is one prerequisite, that each square can be explored just in time.

Lemma 2 *If the BFS is slowed down by a constant factor σ, then each square can be explored in time.*

Proof: A simple partition will not be partitioned by exploration messages because too many border nodes are found. The only possibility to subdivide such a partition occurs if a shoreline sends a "notify"-message to an entry node and this message "starves". Then there may be not enough time to explore the $g \times g$ square and thus it is subdivided. We now prove that this is not the case.

When the shoreline enters an orthogonally neighboring frame A of size $g \times g$, at the entry node v (see Figure 5), then the distance to a node u on the diagonally neighboring frame B is at least $||u - v||_1 \geq \frac{g}{3} - \frac{g}{\gamma}$, because the shoreline comes from a neighboring sub-frame C with side length of at least $\frac{g}{3}$ and in this frame a detour around a barrier is restricted to $\frac{g}{\gamma}$. The time for notification of an entry node v in B is therefore $\frac{g}{3} - \frac{g}{\gamma}$, the exploration of B takes $8(1 + \frac{1}{\gamma})3g$ time steps such that the "close"-message succeeds. The algorithm chooses a frame size of at most $3g$. The exploration is completed before the shoreline enters B for a constant slow-down factor σ.

$$\begin{aligned}
\sigma \cdot \left(\frac{g}{3} - \frac{g}{\gamma}\right) &\geq \frac{g}{3} - \frac{g}{\gamma} + 12\left(1 + \frac{1}{\gamma}\right)3g \\
\iff \qquad \sigma - 1 &\geq 36\frac{\left(1 + \frac{1}{\gamma}\right)g}{\left(\frac{g}{3} - \frac{g}{\gamma}\right)} \\
\iff \qquad \sigma &\geq 108\frac{\gamma + 1}{\gamma - 3} + 1 \geq 540 \quad \text{for } \gamma \geq 4.
\end{aligned}$$

■

This exploration is always done in time, since the size of the frame is chosen such that there is enough time to succeed (in the worst case only one unit square is explored). Since the shoreline proceeds on the frames of the simple partitions, we investigate the length of shortest paths on such partitions.

Lemma 3 *Given a $g \times g$ mesh with frame nodes u and v. Let P be the shortest barrier-free path connecting u and v. A perfect subdivision of the mesh contains a path P' with length $|P'| \leq 2|P|$.*

Proof: We observe that the recursive subdivision produces a fine-grained grid in the proximity of barriers, such that all nodes adjacent to a barrier are part of the grid. If a part of P goes along a barrier,

then this part is also contained in the grid. If a part of P goes through "open space" and is not part of the grid, then it has to be replaced by a path on a frame. The largest detour is caused if the original path enters the frame on the center of one side and leaves it on the center of the opposite side, which enlarges the original path by a factor of two. ■

For each square we allow a small detour depending on the size. It is crucial to know the maximum number of squares of a given size which is described by the following lemma.

Lemma 4 *Let $|P|$ the length of path P and $\ell_i(P)$ the number of squares of side length 3^i that are intersected by the path P. Then $\ell_i(P) \leq 2\frac{|P|}{3^i} + 4$.*

Proof: In order to determine the number of squares we consider the following model: A traveler walks from u to v on the path P. There is a charge for every square. If he enters a square, he has to pay this charge. But, as a reasonable compensation, he receives a mileage allowance. If we determine the mileage allowance such that he is guaranteed not to become insolvent, then we obtain an estimation for the maximum number of squares he can visit.

We consider squares of side length 3^i (level-i squares). Every square costs one dollar. Therefore, we fix the mileage allowance at a rate of two dollars per side length. The traveler starts with 4 dollars beforehand. In order to focus on the relevant aspects, we look at the borders of the squares and especially on the corners. While moving in the plane, we limit the scope to a quadratic area with the corner of the level-i squares in the center. This is depicted by the dashed square in Figure 6 a. If he touches the border, the scope changes (only vertically or horizontally). Visited squares are marked (shaded). If a visited square is out of the scope, we ignore the mark (see Figure 6 b). Thus, one possibly has to pay twice for a square. If one's financial strength allows to pay this extra cost, then it will suffice in either case.

Figure 7 shows the four cases for possible arrangements of marked (i.e. visited) and unmarked squares (within the scope). Other arrangements are equivalent to one of these cases because of symmetry. Transitions between these cases occur, if one crosses the horizontal or vertical borders of the scope. One can easily see, that transitions lead either to case ① $\rightarrow$ A), where one square is already visited, or to case ② ($\rightarrow$ B), where two squares are visited. In the figure the transitions into these cases are written beside the horizontal and vertical borders of the scope.

The amount of money one must possess at some point is also specified in the figure. In case ①, e.g., there is a joint corner of three unmarked squares. Thus, one needs three dollars when reaching this point. At the border of the scope one needs only two dollars, because on the way to the center one receives the mileage allowance. Note, that the distance from the border of the scope to the center is half the side length (distances are measured with respect to the Manhattan metric), for which a mileage allowance of one dollar is paid. One can see that the specified financial requirements agree with a mileage allowance of two dollars per side length ($2/3^i$) in all the four cases.

A fifth case applies for the initial situation where we observe no visited square within the scope. For that case a seed capital of four dollars is required (to pay the charge for four unmarked squares).

Altogether, for a path of length $|P|$ we need an amount of $2\frac{|P|}{3^i} + 4$ dollars. ■

After the exploration a simple partition in a $g \times g$ frame contains a detour of at most $g/\gamma(t)$, where $\gamma(t)$ is defined as a function of the time of exploration t. So, $\gamma(t)$ is not known until the exploration starts. For the time analysis we need a bound for this start time. The algorithm always starts with small squares. So the exploration of a square of size 3^i is not started until smaller squares up to size 3^{i-2} are reached by the shoreline, which happens after $\sigma(3^{i-1} - 1)$ time steps. For a fraction of $(1 - \varepsilon), \varepsilon > 0$ we get a better bound, which is stated in the following lemma.

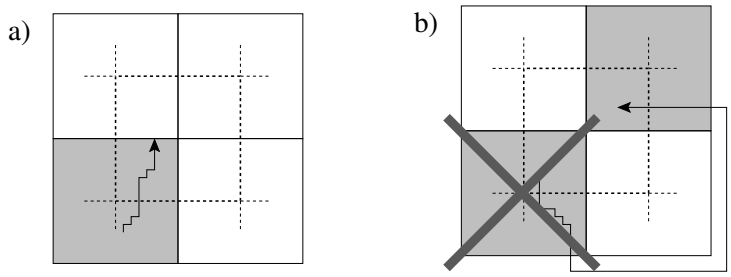

Figure 6: Illustrations for the proof of Lemma 4. a) The transition between squares (dashed lines) takes place in a quadratic area, called the *scope* (shaded square). b) This transition cannot occur, because the first mark is ignored when the scope changes.

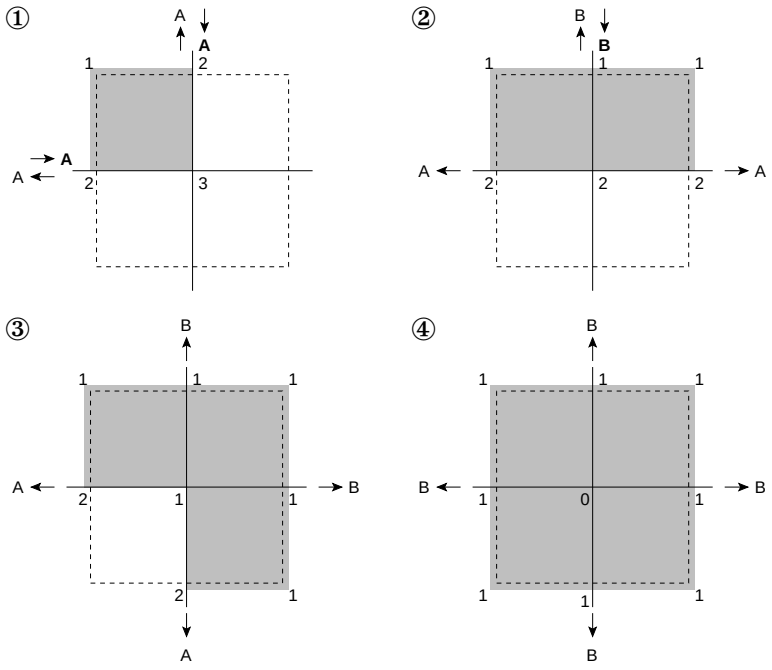

Figure 7: Illustrations for the proof of Lemma 4. Marked squares, which have been visited before, are shaded. Transitions are denoted with arrows. The numbers show the costs at certain positions.

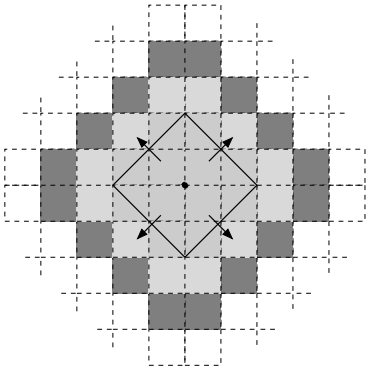

Figure 8: Illustration for the proof of Lemma 5: The black diamond is the shoreline, in the shaded squares the exploration has started

Lemma 5 *Let n_i be the number of squares with side length 3^i. Then, at least $(1-\varepsilon)n_i$ squares are explored after $3^i\sigma\sqrt{\varepsilon n_i}$ time steps.*

Proof: The smaller $\gamma(t)$, the larger is $3^i/\gamma(t)$ the length of the detour. So, the worst case is to place as much squares as near as possible to the source node. The start of the exploration (then $\gamma(t)$ is set) has happened when the shoreline has reached the neighboring squares. If the shoreline has reached a radius of $R := 3^i\, r$ at a time t, then there are at most $2(r+2)(r+3) =: \varepsilon\, n_i$ squares, where the exploration has started or is possibly finished (see Figure 8). For the remaining $(1-\varepsilon)n_i$ squares, the exploration starts after t. The radius R must be at least $2(r+2)(r+3) \geq 2r^2$ to cover $\varepsilon\, n_i$ squares. The shoreline reaches R at time $t = \sigma\, R$. From $\varepsilon\, n_i \geq 2r^2 = 2\left(R/3^i\right)^2 \;\Rightarrow\; R \leq 3^i\sqrt{\varepsilon\, n_i/2}$ follows the time bound.

■

The variable stop criterion $\gamma(t)$ results in a variable detour and a deformation of the BFS tree. The shoreline needs longer to traverse a square, but the time analysis will show that this overhead, which occurs in the beginning, is dominated by the time needed to traverse those squares that are explored later. So, the constant progress of the shoreline is not affected.

The shoreline uses only the borders of simple partitions and approximates the shortest path by following traversal paths on simple partitions of frames. These partitions contain a detour depending on the frame size. The frames that are intersected by the shortest path are also visited by the shoreline. To obtain a bound on the overall path length we consider all frame sizes g, count the number of frames of a certain size (Lemma 4) and add the detour of $g/\gamma(t)$ per frame.

Theorem 2 *Let P be the shortest path with length $|P|$ connecting s and t. The algorithm finds a path P' of length $\mathcal{O}(|P|)$.*

Proof: The path P visits $\ell_i(P)$ frames of size 3^i. We know from Lemma 4 that $\ell_i(P) \leq 2\frac{|P|}{3^i} + 4$. The subdivision of the search area contains a path P' which can be constructed as follows: Determine a shortest path in the perfect subdivision and then add the detour around the barriers inside the frames. For each explored frame of size 3^i this adds a detour of length at most $3^i/\gamma(t)$, which depends on the time of the exploration. We have to consider all frames up to size $|P|$. So the overall length of P' is

given by the following term:

$$\begin{aligned}|P'| &\leq 2|P| + \sum_{i=1}^{\log|P|} \ell_i(P)\,\frac{3^i}{\gamma(t)} \\ &= 2|P| + \sum_{i=1}^{\log|P|} (1-\varepsilon)\cdot\ell_i(P)\,\frac{3^i}{\gamma(t)} + \varepsilon\cdot\ell_i(P)\,\frac{3^i}{\gamma(t)}\end{aligned}$$

We choose $\gamma(t) = \log(t)$. From Lemma 5 we get a bound of $t \geq 3^i\sigma\sqrt{\varepsilon n_i}$ for a fraction of $1-\varepsilon$ of the squares. For the remaining squares $t \geq 3^i \Rightarrow \log(t) \geq i$.

$$|P'| \leq 2|P| + \sum_{i=1}^{\log|P|} (1-\varepsilon)\cdot\ell_i(P)\,\frac{3^i}{\frac{1}{2}\log(3^i\varepsilon\ell_i(P))} + \varepsilon\cdot\ell_i(P)\,\frac{3^i}{i}$$

As $x/\log(cx)$ grows monotonically for $x > e/c$,
from the bound for $\ell_i(P)$ from Lemma 4 follows:

$$\begin{aligned}&\leq 2|P| + \sum_{i=1}^{\log|P|}\left((1-\varepsilon)\cdot\frac{\left(2\frac{|P|}{3^i}+4\right)3^i}{\frac{1}{2}\log\left(3^i\sigma\,\varepsilon\left(2\frac{|P|}{3^i}+4\right)\right)} + \varepsilon\cdot\left(2\frac{|P|}{3^i}+4\right)\cdot\frac{3^i}{i}\right) \\ &\leq 2|P| + \sum_{i=1}^{\log|P|}\left((1-\varepsilon)\cdot\frac{2|P|+4\cdot3^i}{\frac{1}{2}\log(2\sigma\,\varepsilon|P|)} + \varepsilon\cdot\frac{2|P|+4\cdot3^i}{i}\right) \\ &\leq 2|P| + (1-\varepsilon)\cdot\left(\frac{4|P|\log|P| + 8\sum_{i=1}^{\log|P|}3^i}{\log(2\sigma\,\varepsilon|P|)}\right) + \varepsilon\sum_{i=1}^{\log|P|}\left(\frac{2|P|+4\cdot3^i}{i}\right) \\ &\leq 2|P| + (1-\varepsilon)\cdot\left(\frac{4|P|\log|P| + 12|P|}{\log(2\sigma\,\varepsilon|P|)}\right) + \varepsilon\cdot(2|P|\log\log|P| + 8|P|)\end{aligned}$$

Choose $\varepsilon = 1/\log|P|; (1-\varepsilon) \leq 1$

$$\leq 2|P| + \frac{4|P|\log|P| + 12|P|}{\log(2\sigma\,|P|/\log|P|)} + \frac{2|P|\log\log|P| + 8|P|}{\log|P|}$$

$x/\log_3 x > \sqrt{x}$ for $x \geq 1$

$$\begin{aligned}&\leq 2|P| + 16|P|\,\frac{\log|P|}{\log(2\sigma) + \frac{1}{2}\log|P|} + 10|P|\,\frac{\log\log|P|}{\log|P|} \\ &\leq 44|P|\end{aligned}$$

■

The shoreline (BFS) is slowed down by a constant factor σ and uses frames which are explored just-in-time providing a constant factor approximation of the shortest path.

Corollary 2 *Let d be the length of the shortest path connecting s and t. The algorithm finds a path in time $\mathcal{O}(d)$.*

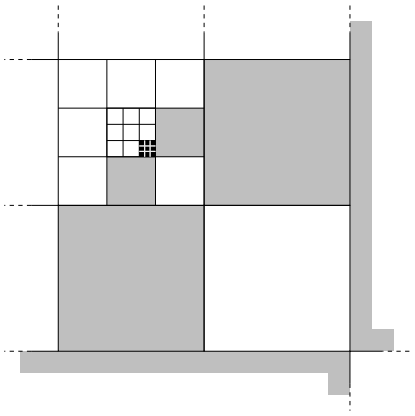

Figure 9: Worst case for neighbor-induced subdivision: After the subdivision of the black square the shaded squares have to be subdivided, too.

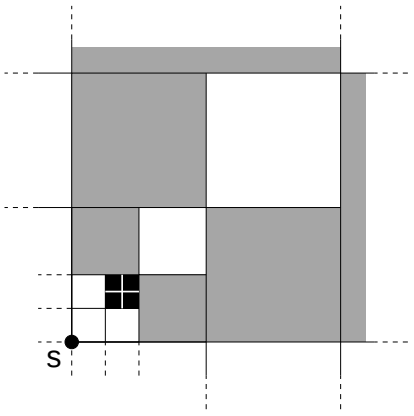

Figure 10: Domino effect in a 2×2-subdivision. A subdivision of the black square triggers subdivisions in the shaded squares.

4.2 Traffic

The traffic depends on the number of quadratic subnetworks (frames) that are explored and subdivided by the algorithm and that constitute the search area. A subdivision of a square is caused by barriers inside the square and by subdivisions in neighboring squares because of the Subdivision Rule (see Section 3.2.1). Therefore, we distinguish between *barrier-induced subdivisions* and *neighbor-induced subdivisions*. A barrier-induced subdivision occurs, if at least g/γ barrier nodes are inside the square (whether they are found or not). All other subdivisions are called neighbor-induced.

A problem is that neighbor-induced subdivisions can trigger a cascade of further subdivisions. Especially the 2×2 subdivision suffers from this domino effect in the initial subdivision of the enlarged search area (see Figure 10), where a small barrier is responsible for subdividing large squares. If a 3×3-subdivision is applied in this case, the domino effect can be prevented. A domino effect can only occur, if a square is subdivided recursively and a barrier-induced subdivision of an inner square triggers recursive subdivisions up to the top-level square (see Figure 9). In this case we can shift the responsibility for the subdivisions from the barriers in the small inner square to the barriers that have caused the recursive subdivision. So, we quote the barriers for the cost of subdivisions of neighboring squares—even if they are not carried out immediately. This is expressed by the following rule:

Payment Rule: A barrier-induced subdivision of a $g \times g$ square pays for

- subdivisions of four neighboring $3g \times 3g$ squares (Rule 1)
- the subdivision of one neighboring $9g \times 9g$ squares (Rule 2)

A barrier that causes a subdivision "pays" for the (barrier-induced) subdivision of the current square as well as for the (neighbor-induced) subdivision of the neighboring squares. This extra cost adds a constant factor to the traffic for exploring a single square. It can be regarded as *amortized traffic*.

Lemma 6 *Let T be the traffic produced by a barrier-induced subdivision of a $g \times g$ square. Then this subdivision causes amortized traffic of at most $14T$.*

Proof: The traffic of a barrier-induced subdivision is not only quoted to the barriers inside the square, but also to neighboring squares according to the Payment Rule.

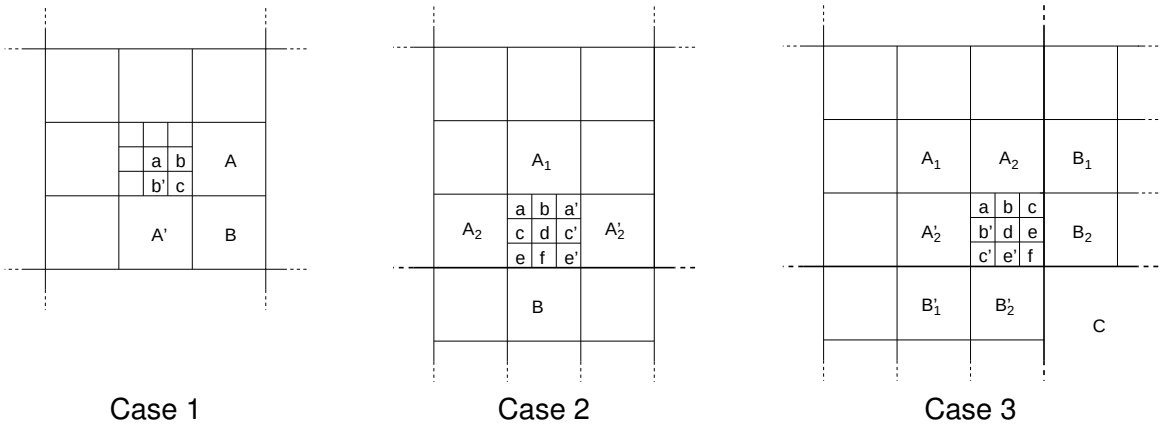

Figure 11: Illustrations for the proof of Lemma 6. Symmetric cases are indicated by apostrophes.

We consider all cases where neighbor-induced subdivisions can occur. There are three major cases to select a square out of a 3×3-subdivision for further subdivision: The middle square (case 1), a square at the side (case 2) and a square at the corner (case 3) as shown in Figure 11. If the small square "b" in case 1 in the figure is further subdivided, then the Subdivision Rule requires a subdivision of square "A". The subdivision of this orthogonally neighboring square is already paid according to Payment Rule 1, because the middle square (containing a,b,c) is already subdivided. This triggers a further subdivision on a higher level, but this case can be reduced to case 1 b. This case and the other cases are shown in the following table:

Subdivided square		Induced subdivsion	Payment rule	Further subdivisions
case 1	a)	none	–	–
	b)	B	1	case 1 b
	c)	B, B'	1	case 1 b
case 2	a)	A_1, A_2	1	A_1: none; A_2: case 1 b or 3 f
	b)	A_1	1	none
	c)	A_2	1	case 1 b or 3 f
	d)	none	–	–
	e)	A_2, B	1	A_2: case 1 b or 3 f; B: none
	f)	B	1	none
case 3	a)	A_2, A'_2	1	none
	b)	A_2	1	none
	c)	A_2, B_2	1	A_2: none; B_2: case 1 b or 3 f
	d)	none	–	–
	e)	B_2	1	$B_2 \to C \to$ case 1 b or 3 f
	f)	B_2, B'_2, C	1, 2	$B_2, B'_2 \to C \to$ case 1 b or 3 f

We can see that in all these cases the induced subdivisions are paid. Other cases are symmetric and can be reduced to the ones shown in the table. Note, that in case 2 and in case 3 the neighboring squares containing B or B_1 and B_2, respectively, must have been subdivided, because of Subdivison Rule 1. ■

Now, as we know the induced traffic by a single square, we are able to analyze the overall traffic.

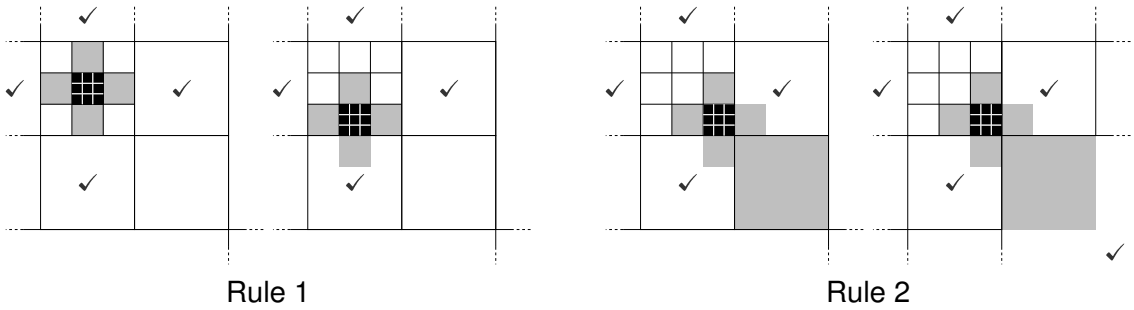

Figure 12: Payment rules for barriers that cause a subdivision. A subdivision of the black square pays also the subdivision of the shaded squares. Marked squares (✓) are already paid.

Theorem 3 *The algorithm produces traffic* $\mathcal{O}(d + p \log^2 d)$.

Proof: We consider the traffic caused by the exploration of $g \times g$ squares. The pure exploration cost of such square is denoted by $E(g) := \eta(1 + \frac{1}{\gamma})4\,g$, where η is the number of visits ($\eta = 7$ because of 5 exploration rounds, the shoreline and stop messages of the continuous ring search), $4\,g$ the length of the sides and $(1 + \frac{1}{\gamma})$ the allowed detour per square.

Assume that there are p border nodes in total. A square is subdivided, if it contains g/γ or more border nodes. Then we have at most $\frac{p}{g/\gamma}$ squares that have to be subdivided into nine sub-squares (barrier-induced subdivision). Each $\frac{g}{3} \times \frac{g}{3}$ sub-square causes the following exploration costs: The exploration of the 9 sub-squares costs $9 \cdot E(\frac{g}{3})$. So the amortized traffic amounts to $14 \cdot 9 \cdot E(\frac{g}{3})$ (Lemma 6). So, the traffic for a square of size $g \times g$ with subdivision can be upper-bounded as follows.

$$\begin{aligned}
\mathbf{Tr}(g \times g) &\leq E(g) + \sum_{i=1}^{\log g} \frac{p \cdot \gamma(t)}{3^i}\, 126\, E(3^{i-1}) \\
&\leq \eta \left(1 + \frac{1}{\gamma(t)}\right) 4\,g + \sum_{i=1}^{\log g} \frac{p \cdot \gamma(t)}{3^i}\, 126 \cdot \eta \left(1 + \frac{1}{\gamma(t)}\right) 4 \cdot 3^{i-1} \\
&\leq 8\,\eta \cdot g + p \log g\,(\gamma(t) + 1)\, 168\, r \\
&\leq 168\,\eta\,(g + p \log g\,(\log(t) + 1))
\end{aligned}$$

Note, that $\gamma(t) = \log(t)$ depends on the time of exploration and therefore the traffic is maximized with a maximum choice of t, which is $\sigma|P'| = \mathcal{O}(d)$, where P' is the approximation of the shortest path found by the BFS.

From the amortized traffic $\mathbf{Tr}(g \times g)$ for a $g \times g$ square, which includes further subdivisions, we obtain the overall traffic for all the squares in the final search area. The maximum radius of continuous ring search is given by $d' := \frac{\sigma+1}{\sigma-1}|P'| = \mathcal{O}(d)$. So the frames constructed by the Fast Exploration Algorithm grow up to a side length of $3^{\lceil \log d' \rceil}$.

The search starts with four squares of size 3×3 (one square in each quadrant and the source node in the origin, see Figure 3). In the next step this area is increased to four squares of size 9×9 etc. Thus, in the i-th step, the area to be explored has a radius of 3^i (with respect to the L_∞-norm). It consists of four squares of size $3^i \times 3^i$ which are subdivided into squares of side length 3^{i-1}. So, in the i-th step there are 8 new squares of side length 3^{i-1} in each quadrant, which have to be explored. As the explored squares of the previous step are not visited again, we have to count only the new

border nodes in each step: Let p_i be the number of border nodes v with $3^{i-1} < ||v-s||_\infty \leq 3^i$. Then, $\sum_i p_i = p$. Note, that $\mathbf{Tr}(g \times g)$ already contains the cost of further subdivisions.

$$\begin{aligned}
\mathbf{Tr} &\leq 4\,\mathbf{Tr}(3\times 3) \;+\; 4 \sum_{i=2}^{\lceil \log d' \rceil} 8\,\mathbf{Tr}(3^i \times 3^i) \\
&\leq 4c\,(3 + p_1 \log h) \;+\; 32 \sum_{i=2}^{(\log d')+1} c\left(3^i + p_i \log 3^i \log h\right) \\
&\leq 32c \sum_{i=1}^{(\log d')+1} \left(3^i + i\,p_i\,\log h\right) \\
&\leq 144c\,d' + 32c \log d \sum_{i=0}^{(\log d')+1} i\,p_i \\
&\leq 144c\,d' + 32c \log d\,(\log d' + 1)\,p \\
&= \mathcal{O}(d + p \log^2 d)
\end{aligned}$$

■

Corollary 3 *The JITE Algorithm has a constant competitive time ratio and a comparative traffic ratio of $\mathcal{O}(\log^2 d)$. It has a combined comparative ratio of $\mathcal{O}(\log^2 d)$.*

5 Conclusions and Open Problems

Conclusions In this paper we solve the open problem of efficient online routing in meshes with faulty nodes. We present an algorithm which can route a message asymptotically as fast as a the fastest algorithm and (up to a term of $\mathcal{O}(\log^2 d)$) with only as many messages as the number of faulty nodes obstructing the messages plus the minimal path length. This considerably improves the known factors of $\mathcal{O}(\sqrt{d})$ [20] and more previously of $\tilde{O}(d^{\sqrt{\frac{\log\log d}{\log d}}})$ [21].

This is achieved by the JITE Algorithm combining several techniques. First we use a continuously expanding search area and establish an adaptive grid of frames which is denser when many barriers are around. On this grid a slowly proceeding shoreline simulates a flooding mechanism. This shoreline triggers the Just-In-Time Exploration (JITE) of new frames that are used by the shoreline. The artful combination of these techniques lead to an algorithm which needs time $\mathcal{O}(d)$ and traffic $\mathcal{O}(d + p \log^2 d)$ where d denotes the length of the shortest path and p denotes the number of border nodes being adjacent to faulty nodes.

Open problems This gives rise to the open question whether these bounds are tight, whether there is a small trade-off between time and traffic. The routing time is delayed by a large constant factor. It seems achievable to decrease this factor without an asymptotic increase of the traffic. However, it is not clear how. Another chance of improvement could be the use of randomized algorithms, which for many other problems outperform deterministic online algorithms.

A straight-forward generalization of this problem are three-dimensional meshes with faulty nodes. The JITE Algorithm, however, in its straight-forward generalization causes a significant increase in traffic. So the question for efficient online routing in higher dimensions is wide open.

References

[1] Dana Angluin, Jeffery Westbrook, and Wenhong Zhu. Robot navigation with distance queries. *SIAM Journal on Computing*, 30(1):110–144, 2000.

[2] Piotr Berman. On-line searching and navigation. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *LNCS*, pages 232–241. Springer, 1998.

[3] Avrim Blum, Prabhakar Raghavan, and Baruch Schieber. Navigating in unfamiliar geometric terrain. *SIAM Journal on Computing*, 26:110–137, 1997.

[4] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[5] Prosenjit Bose and Pat Morin. Competitive online routing in geometric graphs. *Theoretical Computer Science*, 324(2-3):273–288, September 2004.

[6] Prosenjit Bose and Pat Morin. Online routing in triangulations. *SIAM Journal on Computing*, 33(4):937–951, May 2004.

[7] Prosenjit Bose, Pat Morin, Ivan Stojmenovic, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[8] R. J. Cole, B. M. Maggs, and R. K. Sitaraman. Reconfiguring Arrays with Faults Part I: Worst-case Faults. *SIAM Journal on Computing*, 26(16):1581–1611, 1997.

[9] D. B. Johnson and D. A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In T. Imielinski and H. Korth, editors, *Mobile Computing*, pages 152–181. Kluwer Academic Publishers, 1996.

[10] Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300–317, 2000.

[11] Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia. Compass routing on geometric networks. In *Proc. 11th Canadian Conference on Computational Geometry*, pages 51–54, 1999.

[12] F. Kuhn, R. Wattenhofer, and A. Zollinger. Asymptotically optimal geometric mobile ad-hoc routing. In *Proc. of the 6th Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 24–33, 2002.

[13] Caro Lucas. Comments on "dynamic path planning for a mobile automation with limited information on the environment". *IEEE Transactions on Automatic Control*, 33(5):511, May 1988.

[14] Vladimir J. Lumelsky. Algorithmic and complexity issues of robot motion in an uncertain environment. *J. Complex.*, 3(2):146–182, 1987.

[15] Vladimir J. Lumelsky and Alexander A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[16] M. Mauve, J. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network Magazine*, 15(6):30–39, November 2001.

[17] Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. In *Proc. of the 16th Int. Colloq. on Automata, Languages, and Programming (ICALP'89)*, pages 610–620, 1989.

[18] C. E. Perkins, E. M. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. IETF RFC 3561, July 2003.

[19] N. Rao, S. Kareti, W. Shi, and S. Iyenagar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms, 1993.

[20] Stefan Rührup and Christian Schindelhauer. Competitive time and traffic analysis of position-based routing using a cell structure. In *Proc. of the 5th IEEE Int. Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN'05)*, page 248, 2005.

[21] Stefan Rührup and Christian Schindelhauer. Online routing in faulty mesh networks with sub-linear comparative time and traffic ratio. In *Proc. of the 13th European Symposium on Algorithms (ESA'05), LNCS 3669*, pages 23–34. Springer-Verlag, 2005.

[22] Jie Wu. Fault-tolerant adaptive and minimal routing in mesh-connected multicomputers using extended safety levels. *IEEE Transactions on Parallel and Distributed Systems*, 11:149–159, February 2000.

[23] Jie Wu and Zhen Jiang. Extended minimal routing in 2-d meshes with faulty blocks. In *Proc. of the 1st Intl. Workshop on Assurance in Distributed Systems and Applications*, pages 49–55, 2002.

[24] L. Zakrevski and M. Karpovsky. Fault-tolerant message routing for multiprocessors. In *Parallel and Distributed Processing*, pages 714–730. Springer, 1998.