Position-based Routing Strategies

Dissertationsschrift

von

Stefan Rührup

Fakultät für Elektrotechnik, Informatik und Mathematik

Universität Paderborn

Acknowledgments

This work would not have been possible without the understanding of my friends, the encouragement of my parents and the support of all the colleagues with whom I worked together. I have to thank them all and I am especially indebted to the following people.

I would like to sincerely thank my advisor Friedhelm Meyer auf der Heide for giving me the opportunity to choose the direction of my research, for his optimism and his encouragement. I am also grateful to my co-advisor Christian Schindelhauer for the good cooperation in research and teaching, for having patience with me in numerous discussions and for his invaluable help. I also thank Inés Bebea González for burning the midnight oil when working on our research project and the good cooperation despite the thousand miles between Paderborn and Madrid. ¡Muchas gracias!

Special thanks to Silke Geisen and Peter Birkner who didn't shrink away from reading parts of this work in a preliminary state. Their comments helped a lot in improving its readability.

Paderborn, July 2006 Stefan Rührup

Contents

Notation

Frequently Used Variables

B	Barrier, a set of faulty nodes
d	distance of the shortest (barrier-free) path between source and target
g	side length of a quadratic sub-network (square, frame)
γ, $\gamma(t)$	factor, which bounds the detour in a frame (time-dependent)
ℓ	side length of a cell
p	perimeter size (see Section 4.1.1)
P	a path in the network
r	radius (transmission radius or search depth)
$\mathcal{R}_t$	Competitive time ratio (see Section 4.2.1)
$\mathcal{R}_{Tr}$	Comparative traffic ratio (see Section 4.2.2)
$\mathcal{R}_c$	Combined comparative ratio
s	source node
σ	constant factor, by which the search is slowed down
t	target node; time
time	number of time steps
Tr	traffic, i.e. total number of messages

Asymptotic Growth of Functions [Ste01]

$\mathcal{O}(n)$ $\quad f(n) = \mathcal{O}(g(n)) :\Leftrightarrow \limsup_{n\to\infty} \frac{|f(n)|}{|g(n)|} < \infty$

$\Omega(n)$ $\quad f(n) = \Omega(g(n)) :\Leftrightarrow \liminf_{n\to\infty} \frac{|f(n)|}{|g(n)|} > 0$

$\Theta(n)$ $\quad f(n) = \Theta(g(n)) :\Leftrightarrow f(n) = \mathcal{O}(g(n)) \wedge f(n) = \Omega(g(n))$

$o(n)$ $\quad f(n) = o(g(n)) \;:\Leftrightarrow \lim_{n\to\infty} \frac{|f(n)|}{|g(n)|} = 0$

$\omega(n)$ $\quad f(n) = \omega(g(n)) :\Leftrightarrow \lim_{n\to\infty} \frac{|f(n)|}{|g(n)|} = \infty$

Further Notations

$\log n$	the binary logarithm $\log_2(n)$
$\log^k n$	abbreviated notation for $(\log_2(n))^k$
$\lfloor x \rfloor$	floor function (the largest integer less than or equal to x)
$\lceil x \rceil$	ceiling function (the smallest integer greater than or equal to x)

1

Introduction

Position-based routing is the task of delivering a message to a specified position in a network. Instead of using routing tables and network addresses, the routing decisions are made on the basis of the current position and the position of the destination. First approaches for position-based routing were developed in the 1980s for fixed interconnection networks [Fin87] and packet radio networks [TK84, HL86]. The first algorithms were greedy strategies that always forward a packet to a neighbor that is closer to the target than oneself. These greedy strategies have the disadvantage that they fail in case of a local minimum, i.e. at some point where no further progress is possible. In wireless networks, local minima result from void regions that cannot be bridged because of restricted transmission ranges.

In the late 1990s the first position-based routing strategies were developed that guarantee message delivery by constructing alternate paths around local minima. Today we are facing a plethora of different algorithms that can be divided into two categories: single-path strategies and multi-path strategies. Single-path strategies try to reach the target using only one path. This is traffic-efficient, i.e. the total number of messages is normally small, but this path can contain a large detour if local minima have to be circumvented. Multi-path strategies construct many redundant paths to increase the chances of not getting into a local minimum. The extreme case of multi-path routing is flooding. Multi-path strategies need more traffic a priori, but can decrease the time for message delivery.

Most position-based routing algorithms belong to the class of single-path strategies. Over the last years the robustness of such strategies under non-idealized communication models has drawn the attention of the research community, whereas multi-path strategies were mainly developed for geocasting (i.e. multi-casting to all nodes in a geographic region). However, most multi-path strategies still rely on flooding, even though it can be restricted to a geographic region. This raises the following question: Are there efficient multi-path strategies beyond flooding that require less traffic?

This question is answered by this thesis: the Just-In-Time-Exploration (JITE) algorithm, which is presented in Chapter 4, delivers a message asymptotically as fast as a flooding algorithm, but with much less traffic. More precisely, it matches the asymp-

totic lower bound for time and approaches the lower bound for traffic up to a polylogarithmic factor. Known strategies fail to approach both lower bounds at the same time: single-path algorithms need much time in complicated scenarios, whereas flooding algorithms always produce a large traffic overhead.

In order to study such algorithms, we consider the model of a mesh network containing faulty nodes, which are not known in advance. This model can be seen as an abstraction from the graph theoretical model of a wireless network. The faulty nodes in the mesh network may form faulty blocks of arbitrary shape, which may be as complicated as a maze. We call these faulty blocks "barriers" as they obstruct the network communication. Neither the number nor the location of the faulty nodes are known in advance. Therefore, every decision has to be made *online*, i.e. without global knowledge. The task of delivering a message in such a network is a typical online problem.

Online problems are usually analyzed under comparative measures. For the position-based routing problem we compare the performance of an algorithm with the lower bounds for time and traffic: No algorithm can be faster than following the shortest path. Therefore, we consider the maximum ratio of the time an algorithm needs (which is equivalent to the generated path length) and the length of the shortest path. This ratio is called the *competitive ratio* in the context of online algorithms. The online lower bound for traffic depends on the barriers in the scenario: In the worst case, every online algorithm has to traverse all the barriers, regardless whether this is done sequentially or in parallel. Thus, the lower bound for traffic is the length of the shortest path and the perimeters of the barriers. The *comparative ratio* for traffic is the maximum ratio of the traffic produced by an algorithm and the lower bound. Under these measures the disadvantages of flooding algorithms and single-path strategies can be expressed. They optimize time at the expense of traffic and vice versa. The JITE algorithm uses a combination of several techniques in order to avoid the disadvantages of known approaches. It uses a quadtree-style subdivision of the network that becomes denser in the proximity of the faulty parts. The search for the target is performed by a slow proceeding breadth-first search (BFS) on the borders of the subdivision squares. The subdivision is created just-in-time by an exploration process before the BFS arrives. Since there is no global knowledge of the parallel branches in the BFS tree, a local coordination mechanism is needed to prevent repeated exploration of the same area. The asymptotically optimal time can be achieved because the BFS is slowed down by a constant factor and the subdivision contains a constant-factor approximation of the shortest path. This algorithm is described and analyzed in Chapter 4.

Chapter 3 shows an application of the mesh network model. If we map the nodes and communication links of a wireless network to cells of a grid subdivision, we see the similarity to the faulty mesh network: cells which are covered by links can be used for routing, whereas the void regions are barrier cells and the equivalent of the faulty nodes in the mesh network model. In a wireless network where the nodes have position information, such a cell structure can be constructed distributedly by a geographic clustering using only local information. This is part of the Cell-based Geographic Forwarding Protocol, which also contains a forwarding strategy that works on the cell structure.

This thesis is organized as follows:

Chapter 2 – Related Work This chapter contains a review of related work, including navigation algorithms that solve similar problems as position-based routing algorithms and were studied in the context of online algorithms and theoretical robotics.

Chapter 3 – Position-based Routing using a Cell-Structure In this chapter the Cell-based Geographic Forwarding Protocol is presented, which is based on a geographic clustering using a cell structure. This cell structure is constructed distributedly by using position information and provides the basis for a single-path forwarding algorithm. It is shown that routing on the cell structure and routing on the original topology of the network are equivalent. The cell structure corresponds to the model of a faulty mesh network, which forms the basis for the analysis of the routing algorithms presented in Chapter 4.

Chapter 4 – Online Routing in Faulty Mesh Networks In this chapter the routing problem in faulty mesh networks is studied. For the analysis of time and traffic, comparative performance measures are defined. Then, lower bounds for time and traffic are shown. After that, two known routing strategies are analyzed: expanding ring search, which is a time-optimal flooding algorithm, and a traffic-optimal single-path strategy. We will see that both time and traffic can be reduced by a combination of both, but this technique is not sufficient to reach the lower time bound. Finally, the JITE algorithm is described and analyzed. This algorithm outperforms the preceding ones.

Parts of this work have been published on conferences. The cell structure used by the Cell-based Geographic Forwarding Protocol and its properties were described in [RS05a]. The performance measures defined in Chapter 4 and an algorithm for the online routing problem in faulty mesh networks were presented in [RS05b]. This algorithm is a predecessor of the JITE algorithm and bases on similar ideas. However, the JITE algorithm [RS06] uses new techniques to achieve an asymptotically optimal time.

2

Related Work

The fundamental goal of the routing problems studied in this work is to reach a target at a specified position. From a source node a message is forwarded through a network which is not known to the source except for the position of the target. In the field of robotics the similar *navigation* problem is studied: a robot has to find a path to a target, which is also specified by its position. The environment is unknown to the robot and may contain obstacles. For both position-based routing and navigation in unknown environments similar strategies were developed and also similar performance measures are used. This chapter gives an overview of related research results, including algorithms for position-based routing and related navigation problems.

2.1 Position-based Routing

Position-based routing (also called geographic routing) is a reactive routing scheme where forwarding decisions are based on geographical positions of the nodes in the network (see [MWH01, GS03, GS04, KW05] for a survey). It is based on the following assumptions:

1. All nodes can determine their own position.
2. All nodes know the positions of their direct neighbors.
3. The source node knows the position of the destination.

One of the first approaches to use position information for routing in networks was proposed in 1987 by Finn [Fin87]. It is called Cartesian routing and uses location information of nodes in a fixed interconnection network for routing packets in the direction of the destination node. A packet is forwarded to a neighboring node, if there is a progress towards the destination. If the routing algorithm solely uses this greedy strategy, the message delivery is only guaranteed if each node in the network has a neighbor that is closer to the target. Finn makes a weaker assumption: If a neighbor closer to the target can always be found within a constant number of hops, then this neighbor can be found by a limited flooding.

In the same decade, Takagi and Kleinrock [TK84] as well as Hou and Li [HL86] described greedy forwarding rules for the wireless communication in packet radio networks, where the participants know their position. These greedy strategies were later adopted for position-based routing in wireless ad hoc networks. The problem of greedy forwarding is that it fails in case of a local minimum. In wireless networks, local minima exist a the border of void regions, which cannot be bridged because of the limited range of the radio transceivers.

Routing strategies that guarantee message delivery, even in the presence of local minima, were developed in the late 1990s [KSU99, BMSU01]. These strategies determine the routing path by traversing the faces of the network graph. This requires a planar graph, therefore the network topology has to be *planarized*. These approaches rely on the assumption that transmission ranges are uniform so that the network can be modeled by a so-called *unit disk graph*: A unit disk graph contains a link between two nodes u and v if and only if $||u - v|| \leq 1$. This definition corresponds to the assumption that there is a connection between two nodes if their distance is smaller than the normalized transmission radius ($r = 1$).

Using face traversals is less efficient than greedy forwarding. Therefore, Karp and Kung [KK00] and Bose et al. [BMSU01] have proposed the idea to combine both strategies. Every time the greedy strategy fails, the traversal of the incident face is started in order to *recover* from the local minimum. This way, the efficient greedy forwarding can be used where it is possible and also the message delivery can be guaranteed.

2.1.1 Greedy forwarding

Greedy forwarding is based on a local decision of a node to forward a packet to a neighboring node that is nearer to the destination. Local decision strategies are:

- MFR (most forward within the transmission radius) [TK84] tries to minimize the number of hops,
- NFP (nearest with forwarding progress) [HL86] tries to minimize energy consumption, and
- Compass routing [KSU99] tries to minimize the Euclidean path length by choosing the node with the minimal angular distance to the direction of the destination.

Figure 2.1 illustrates these strategies.

Since a packet could get stuck in a local minimum, greedy forwarding must be accompanied by a *recovery strategy* in order to guarantee message delivery. Recovery strategies are usually based on face traversal and planarization, which is explained in the following subsections.

2.1.2 Face Traversal

Face traversal is a routing strategy, where a message is forwarded along the faces of a planar sub-graph of the network topology. This idea is based on the *right-hand rule*, a

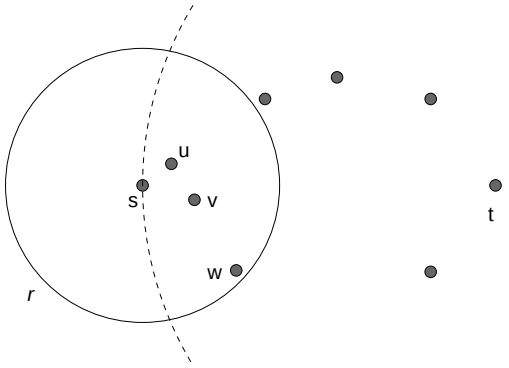


Figure 2.1: *Greedy forwarding: node s can choose u (NFP), v (compass routing) or w (MFR) for the next hop*

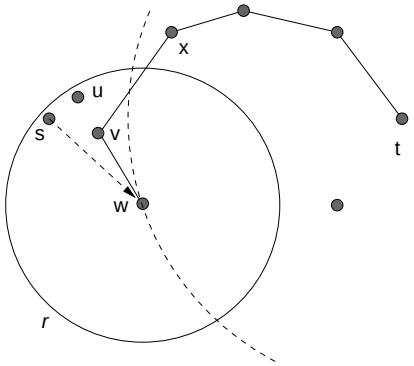


Figure 2.2: *Recovery from a local minimum at w using the right-hand rule*

well-known strategy for traversing a maze (see Section 2.4.3). Instead of following the walls of a maze, the edges of a face (which is a void region in the network and therefore an obstacle for routing) are traversed in clockwise order. The progress towards the target is made by traversing adjacent faces that are intersected by a line segment between source and target (see Figure 2.3). With the aid of position information, the nodes determine the intersection points and know when to switch to the next face. Face traversal algorithms require a planar network topology. Strategies for planarization are described in the next subsection.

Face traversal can be applied as stand-alone routing strategy, but also as recovery strategy in combination with greedy forwarding. Karp and Kung propose a face traversal algorithm, called *perimeter routing* (cf. "face-2" in [BMSU01]) as part of their GPSR protocol [KK00, Kar00]. It traverses the edges of the face incident to the local minimum until a forwarding progress according to the greedy strategy can be made (see Figure 2.2). Another strategy, where the incident face is traversed completely in case of a local minimum ("face routing"), has been proposed by Kranakis et al. (see [KSU99], "compass routing II"; cf. "face-1" in [BMSU01]).

Kuhn et al. have extended the face routing technique in order to guarantee a bounded path length in the worst case. They propose a face routing variant (called OFR) where the complete interior of a face is explored in order to determine the node that minimizes the distance to the destination. Then, at this node the next (adjacent) face toward the destination is chosen. The authors also give an overview of methods to combine greedy and face routing. In [KWZ02] Kuhn et al. propose an adaptive face routing (AFR). The idea is to bound the faces by an elliptic region to prevent detours. If no route to the destination within this region can be found, the region is successively enlarged. The algorithm yields paths with cost $\mathcal{O}(d^2)$ where d denotes the cost of the optimal route measured by the number of hops, the distance or the energy consumption. These approaches were used for the GOAFR+ [KWZZ03] algorithm which uses a combination of

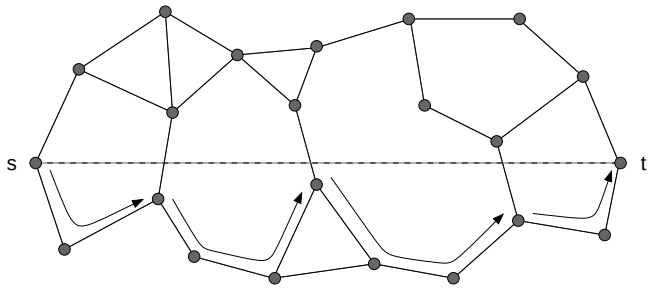


Figure 2.3: *Face routing on a planar network topology*

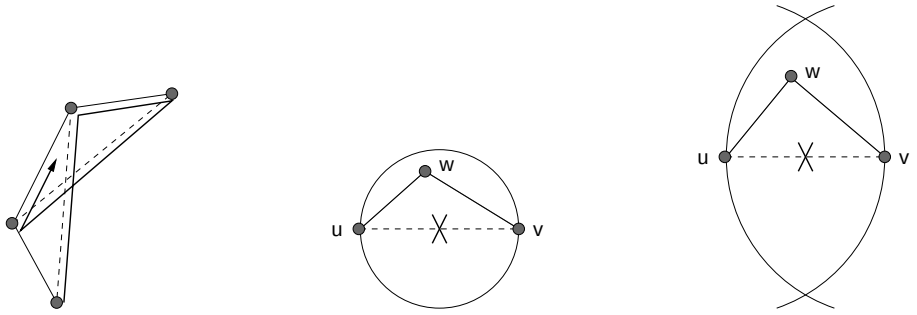
greedy forwarding and face routing and is asymptotically optimal. The authors show a lower bound of $\Omega(d^2)$ with the following construction: the nodes are placed in a way that leaves void regions in between and only some defined paths are left. Then the algorithm has to investigate $\Omega(d)$ paths each of length $\Omega(d)$, but only one of them leads to the target, the others are dead ends. A similar idea for the worst case construction is used to show a lower bound for robot navigation algorithms (see [BRS97] and Section 4.3). In our analysis we take the perimeters of void regions into account, so we can express performance beyond the worst case point of view.

2.1.3 Graph Planarization

The face traversal algorithms require a planar graph[1], because crossing edges may cause a loop when traversing a face as shown in Figure 2.4(a). Therefore, a planar subgraph of the original network topology has to be created by a *planarization* algorithm. There are various planar graphs that can be used for a subgraph construction, e.g. the Gabriel Graph (GG) [GS69] or the Relative Neighborhood Graph (RNG) [Tou80, JT92]. When applying the GG Planarization, an edge (u, v) is eliminated if Thales' circle on (u, v) contains another node w (see Figure 2.4(b)). The RNG Planarization eliminates an edge (u, v) if the intersection of two circles with radius $||v - u||$ centered at u and v contains another node w (see Figure 2.4(c))

Other constructions are based on the Delaunay Triangulation [GGH$^+$01, ALW$^+$03]. All these sub-graphs can be constructed locally by position information and communication in the local neighborhood. These subgraphs have been analyzed with respect to their spanning properties which express how well the subgraph approximates the original topology. A subgraph of a graph is a spanner if the length (Euclidean length, hop distance or power distance) of a path between two nodes is only a constant factor (called the *stretch factor*) larger than the distance of the nodes in the original graph. Bose et al. [BDEK06] have proved length stretch factors of $\Theta(\sqrt{n})$ for the Gabriel Graph and $\Theta(n)$ for the Relative Neighborhood Graph (RNG), i.e. the detours induced by these subgraph constructions are not bounded by a constant. Often the desired properties (e.g. a constant stretch factor) are only valid if a constant minimum distance between the

[1] Here, planarity means that the connectivity graph of the network is a planar embedding.



(a) Traversal loop (b) GG Planarization (c) RNG Planarization

Figure 2.4: *Crossing edges can cause a face traversal loop. The dotted edges are eliminated by planarization.*

nodes is assumed ("civilized graphs" [WL02], $\Omega(1)$-model [KWZ02]). To take advantage of such properties in scenarios with unbounded node density where the minimum distance assumption may be violated, a clustering approach can be applied. Clustering is proposed by Gao et al. [GGH+01] and Alzoubi et al. [ALW+03] to form a backbone network, which connects the clusterheads and has the desired graph properties.

The disadvantage of planarization is that many strategies eliminate advantageous long edges and preserve short ones. This leads to an increased hop distance of the shortest path in the planarized graph compared to the original communication graph. The Cell-based Geographic Forwarding Protocol presented in Chapter 3 uses a cell-based subdivision of the plane to provide each node with a map of its environment and to characterize borders of void regions. The protocol works without canceling links, because the cell-structure provides an implicit planarization.

Fang et al. [FGG04] proposed an algorithm for identifying local minima and void regions (which they call "routing holes") based on the geometric properties of the network topology. The algorithm uses local rules to determine possible local minima. Then the void regions are identified by communication of the nodes at the border of this region. So the traffic used for face traversals in recovery mode is shifted from the routing algorithm to the proactive part of the communication.

Robustness of Planarization

Many local planarization strategies rely on the unit-disk-graph assumption where the transmission ranges are identical and uniform. Barrière et al. [BFNO03] have shown that under certain conditions (all links are bidirectional and the ratio of the minimum and the maximum transmission range is smaller than $\sqrt{2}$) a planarized graph can be constructed with local information by exchanging neighborhood information and applying the Gabriel Graph planarization on a virtual graph. The impact of non-uniform transmission ranges has been studied by Kim et al. [KGKS05b]. Planarization can cause disconnected links or errors in face changes. They developed the Cross-Link Detection

Protocol (CLDP) [KGKS05a], which starts right-hand traversals repeatedly to detect and remove crossing links. This results in a planarized subgraph, on which face routing guarantees message delivery. Another approach for position-based routing that avoids the use of graph planarization is the Greedy Distributed Spanning Tree Routing (GDSTR) [LLM06]. This protocol uses the traversal of spanning trees instead of face routing on a planarized graph in cases where greedy forwarding fails. The distributed construction of these spanning trees needs at most $3D$ rounds, where D is the diameter of the network graph. Both protocols cannot rely on local information only and thus have an increased communication overhead.

2.1.4 Beacon-less routing

The described greedy forwarding and face traversal algorithms rely on the assumption that the positions of neighboring nodes are known. There are also position-based routing protocols that do not need this information. Instead, they use the position data for implicit addressing: A message is broadcasted to the neighboring nodes including position data of the sender and the target. Then the neighbors compete for the forwarding task, where the contention period is determined by the relative positions to the sender. The node that is nearest to the destination may answer first and take over the message. Protocols that work in this way are beacon-less routing (BLR) [HB03, HBBW04], contention-based forwarding (CBF) [FWMH03] and implicit geographic forwarding (IGF) [BHSS03]. Furthermore, the geographic random forwarding (GeRaF) protocol also belongs to this class of protocols [ZR03b, ZR03a]. IGF and CBF work with a greedy strategy only and thus require a high node density to avoid greedy failures. BLR contains also a recovery strategy: in recovery mode the contention period is chosen proportional to the angular distance to the last edge in recovery mode.

These routing strategies show that position-based routing can be performed completely reactive. The advantages of reactive and proactive information exchange are discussed in Section 3.2.

2.1.5 Position-based Multi-path Strategies

Besides greedy forwarding and face routing, there are multi-path strategies that construct several routes to a destination or use multi-casting or flooding to inform all nodes in a specific region. One example is *Restricted directional flooding* [BCSW98]. The goal of this algorithm is to handle outdated position information. From the last known position of the destination node and its maximum velocity a destination region is obtained. Then, the sender transmits a packet to all nodes that lie in the direction of this region in order to flood the whole area in which the destination node lies. This strategy can be used to reduce the overhead of flooding used by reactive ad hoc routing protocols in the route discovery phase (location-aided routing, see [KV00]).

Similar approaches are also used for *geocasting* (multi-casting to a geographic region). Surveys on geocasting protocols are given by Jiang and Camp [JC02], Maihöfer [Mai04] and Stojmenovic [Sto04, Sto06].

Most of these approaches are based on flooding, which is not traffic-efficient though it can be restricted to a geographic region. The JITE algorithm presented in Chapter 4 opens a new direction towards traffic-efficient multi-path strategies.

2.1.6 Geographic Clustering

A grid or mesh based subdivision of the plane is used by several position-based routing protocols (e.g. GRID [LTS01]) where the grid cells are used for geographic clustering, i.e. for each cell there is one elected node that is the gateway for the communication with neighboring clusters. Using only these gateway nodes for forwarding, the remaining nodes can be freed from inefficient communication tasks.

These approaches for geographic clustering are used for constructing redundant routes and efficient multicasting or geocasting. GeoGRID [LLS00] is a geocasting protocol based on GRID that uses flooding for reaching the geocast region. Chang et al. [CCT03] use a subdivision in hexagonal cells and propose a geocasting scheme that works in two phases. In the first phase the message is sent to one node in the destination region. Obstacles regions which are cells without any node are bypassed by flooding. Finally, geocast region is flooded. Camp and Liu [CL03] use the inter-gateway communication to create redundant routes for reaching the geocast region.

The Cell-based Geographic Forwarding Protocol (see Chapter 3) uses an implicit clustering based on a cell structure. The advantage of the cell structure over the planar subgraph constructions is that no links to neighboring nodes are explicitly forbidden. Void regions are characterized by so-called barrier cells, so that a right-hand recovery can be applied that uses the information about these cells. Topology-based rules that work on the edges of the network connectivity graph and that require a prior planarization are not necessary.

2.1.7 Location Services

The third assumption for position-based routing is that the position of the destination node is known. To provide each node with this information is the task of *location services*. The problem of locating the destination node is somehow similar to the route discovery problem in ad hoc network protocols. *Proactive* schemes disseminate routing information before it is needed, whereas *reactive* schemes start the route discovery on demand. Accordingly, a location service may work in a proactive way: each node maintains a location database and distributes its own position throughout the network at regular intervals [BCSW98]. To reduce the update cost, such location databases can also be maintained by a small number of dedicated nodes that act as location servers. The location-aided routing (LAR) protocols by Ko and Vaidya [KV00] work completely reactive. Position information is only used if it is available. If the destination's position is unknown, the network is flooded with route requests.

Li et al. [LJC+00] propose a distributed location service, called Grid Location Service (GLS) that uses a hierarchical geographic subdivision of the plane in order to assign location information to dedicated hosts. Rao et al. [RPSS03] as well as Ratnasamy et

al. [RKY+02] use distributed hash tables to assign location information to dedicated regions in the network, where this information can be looked up by a query which is geographically forwarded to and answered by a node in this region. Rao et al. [RPSS03] also propose an algorithm for establishing virtual coordinates that enable geographic routing if position information is not available.

2.1.8 Related Online Routing Problems

Bose and Morin [BM04b, BM04a] 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 there are no local minima, routing can be done by a greedy strategy as it is used for position-based routing.

2.2 Routing in Faulty Networks

In this work a faulty mesh network is used as an abstract model for studying online routing algorithms (see Figure 2.5). Routing in faulty networks has also been considered as an offline problem. In the field of parallel computing the fault-tolerance of networks is studied, e.g. by Cole et al. [CMS97]. The problem is to construct a routing scheme in order to emulate the original network. Zakrevski and Karpovski [ZK98] investigate the routing problem for two-dimensional meshes. They use a similar model as they consider store-and-forward routing in two-dimensional meshes. 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 [Wu00]. These algorithms use only local information, but the faulty regions in the mesh are assumed to be rectangular blocks. In [WJ02] 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.

2.3 Online Problems and Performance Measures

Routing in unknown networks as well as navigation in an unknown environment can be regarded as *online problems*. Algorithms for such problems have to make decisions without global knowledge and without knowing about the future. Classical online problems are, e.g. task scheduling or paging where the algorithm has to handle requests for processors or memory pages without knowing about future requests. The analysis of these problems led to the framework of the so-called *competitive analysis* [MMS88, BE98]. The basic idea of the competitive analysis is to consider a competition between the online algorithm and the best offline strategy. This competition is carried

out over all instances of the problem. Then, the maximum ratio of the cost of the online algorithm and optimal offline cost gives the so-called *competitive ratio*.

In position-based routing as well as in online navigation, the algorithm has to compete with the shortest path to the target. But there are other performance measures beyond the classical competitive ratio, which have been studied in other fields of research. Navigation problems have been investigated in different research communities, which Angluin et al. [AWZ00] called "the online competitive analysis community" and the "theoretical robotics community". 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 [LS87, AWZ00]. 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 [PY89, BRS97].

The model of a faulty mesh network connects these two lines of research. The connection between the scenarios considered in online searching and this model is obvious: Scenarios with a lower bound on the distance between starting point and target point and with finite obstacle perimeters can be modeled by a mesh network with barriers consisting of faulty nodes. Online routing in such a network and the problem of navigating to a point in an unknown environment are essentially the same, but the network model imposes no restrictions on the parallelism. 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. Moreover, the traffic is compared to the perimeters of the barriers which gives the comparative traffic ratio (see Section 4.2). This ratio expresses the amount of parallelism used by the algorithm.

2.4 Online Navigation and Searching

2.4.1 Navigation

Navigation is the task to find a path in an unknown environment to a target, which is specified by its position (see Figure 2.6). Problems of this type have not only drawn the attention of the robotics community, they were also studied in the field of online algorithms, where the competitive analysis is used to evaluate the performance of algorithms. While in theoretical robotics this performance is expressed by the overall path length in terms of the distance to the target and the perimeter of obstacles, in the competitive analysis the competitive ratio is mainly used. The competitive ratio compares the cost of an algorithm with the cost of the optimal solution. For online searching

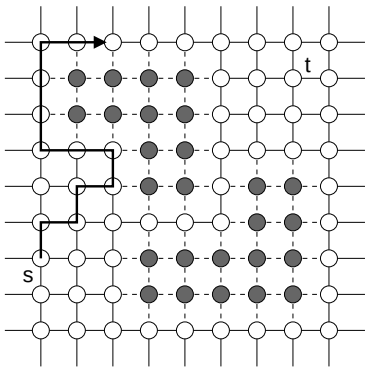


Figure 2.5: *A mesh network with faulty nodes*

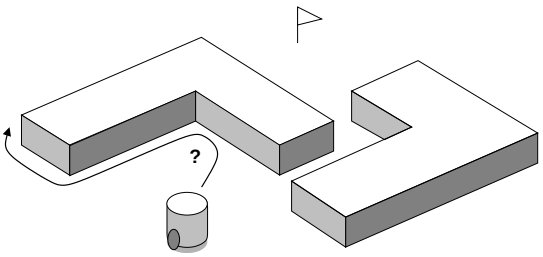


Figure 2.6: *Online navigation: a robot has to reach a target (indicated by the flag) with only touch sensing*

and navigation problems the cost is usually the path-length between source and target.

The competitive analysis of online navigation problems was introduced by Papadimitriou and Yannakakis in 1989 [PY91]. They studied the problem of navigation in two-dimensional scenes with oriented rectangular obstacles. For quadratic obstacles they showed a lower bound of 3/2 on the competitive ratio. They also introduced the *wall problem*, where the goal is to reach a vertical line, and showed a lower bound of $\Omega(\sqrt{d})$ on the competitive ratio for deterministic algorithms. Blum, Raghavan and Schieber [BRS97] later found an algorithm for the wall problem that matches this lower bound. For this problem, randomization can be used to overcome the deterministic lower bound: Berman et al. [BBF+96] presented a randomized algorithm with a competitive ratio of $\mathcal{O}(n^{4/9} \log n)$.

The algorithms community has studied several related problems with various restrictions on the environment or on the obstacles. Blum, Raghavan and Schieber introduced the *room problem*, where the starting point is located on the wall of a quadratic room and the target is the center of the room. The obstacles are also oriented rectangles. In this problem there is always a path of length d. An optimal algorithm with a competitive ratio of $\mathcal{O}(\log d)$ was presented by Bar-Eli et al. [BBFY94].

For more complex obstacles the navigation problem becomes much harder. In the case of arbitrary non-convex obstacles, as they are assumed in theoretical robotics, one can show a linear lower bound on the competitive ratio, i.e. any algorithm exceeds the optimal path length of d by a factor of d. This is shown in [BRS97] by constructing a maze-like structure which consists of several corridors, where only one of them leads to the target (cf. Figure 2.7). The online algorithm has to examine all corridors, whereas the offline solution is to directly take the right corridor leading to the target. For this problem there is no better solution than to perform a complete obstacle traversal. The same considerations lead to the lower bound for traffic for online routing algorithms, which is stated in Section 4.3.

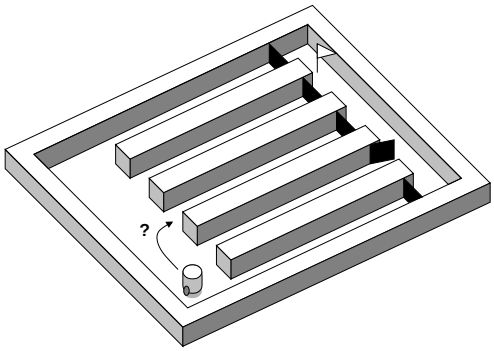


Figure 2.7: *Lower bound scenario for online navigation: A robot (without vision) has to examine all corridors in the worst case.*

Restrictions on the robot

Hemmerling [Hem93] studied the navigation problem for a robot that has neither position nor distance information. When having a compass, the target can be reached in $\mathcal{O}(n^2)$ steps using a rotation counter, where n is the number of vertices of the polygonal obstacles.

Angluin, Westbrook and Zhu [AWZ00] investigated robot navigation under the restriction that the robot can only determine the distance to the target instead of having position information. According to their algorithm, a robot moves towards the target and follows the obstacle boundaries, but it has no range information to determine the point where the obstacle boundary can be left. Instead, the criterion for stopping the boundary traversal is a sequence of local minima and maxima on the obstacle boundary. Like in [LS87] the analysis is focused on the perimeters of the obstacles: As a performance measure they consider the ratio of the excess distance (which is the length of the robot's path minus the straight-line distance) and the sum of the perimeters. In Section 4.2 we define a similar comparative measure for the traffic of online routing algorithms.

2.4.2 Searching and Exploration

Navigation means to reach a target at a position which is known beforehand. If this position is not known, then the whole environment has to be investigated because the target can be placed where the algorithm searches last. Problems of this type are known by the term *searching*. Another related problem is the *exploration* of an unknown environment. Exploration algorithms and searching algorithms have to investigate the whole environment, but for both problems different competitive measures are used. In online searching one usually compares the length of the path the algorithm has found with the length of the shortest path. In contrast, online exploration algorithms have to compete with the shortest tour that covers the complete environment.

The competitive analysis community has studied several problems ranging from

searching for a point on a line to the exploration of graphs. Probably the best known problem is the so-called *cow-path problem*, which is sometimes mentioned as prime example for a competitive analysis: A short-sighted cow searches for a gate in a fence. It can only detect the gate when standing directly in front of it. So it has to walk along the fence, but does not know in which direction. The offline solution is simple and the cost of an offline algorithm is just the distance from the initial position to the gate. The best online solution (for the cow) is the following doubling strategy: walk one step to the right, return to the initial position, move two steps to the left, walk back and then four steps to the right, and so on until the gate is reached. In each iteration the search depth is doubled. So the length of the cow's path is only a constant factor larger than the distance between the starting point and the gate. This strategy is optimal and yields a competitive ratio of 9. This problem can be regarded as a search for a point on a line and is a special case of the search on m concurrent rays, where the cow is standing on a crossroad and has m directions in which it probably finds the goal. Upper and lower bounds for this problem have been shown by Baeza-Yates et al. [BCR93] and Jaillet and Stafford [JS01]. Baeza-Yates et al. describe the Linear Spiral Search algorithm that sequentially visits the m rays and chooses a search depth of $\left(\frac{m}{m-1}\right)^i$ in the i-th iteration. This is a generalization of the doubling strategy for the case of two rays. For this problem, randomization helps to overcome the lower bounds of deterministic algorithms. The randomized SmartCow algorithm of Kao et al. [KRT93] chooses a random initial offset and a random permutation for visiting the m rays. This algorithm is optimal and beats the deterministic strategy: It achieves a competitive ratio of approximately 4.59 for m=2, compared to the competitive ratio of 9 for the deterministic algorithm. The cow-path problem was introduced earlier and independently analyzed by Gal [Gal80] in the context of game theory. Gal showed the optimality of exponential functions, which are used for choosing the search depth in this problem. Choosing exponential functions is not only useful for these kind of search problems. A doubling strategy can be also used, e.g., to limit the traffic overhead of flooding in networks. This is known as *expanding ring search*, which is described in Section 4.4.2. We use expanding ring search as part of the alternating algorithm, which reduces time and traffic overhead at the same time.

The exploration problem, which is related to searching, has been studied by Deng et al. [DKP98] for robots with vision capabilities, by Albers et al. [AKS02] for a robot in a two-dimensional environment, by Kleinberg [Kle94] and Icking et al. [IKKL05] for simple grid polygons, Deng and Papadimitriou [DP99] as well as Albers and Henzinger for graphs [AH00]. Deng and Papadimitriou showed that the exploration problems can be solved efficiently for Eulerian and similar graphs. They introduced the definition of the *deficiency* of a graph, which is the minimum number of edges that have to be added to make the graph Eulerian. They present a lower bound of $\Omega(d^2m)$, where d is the deficiency and m the number of edges in the graph.

A survey of searching and navigation algorithms is given by Berman [Ber98]. Some of the results are also described in [LaV06].

2.4.3 Maze Traversal Algorithms

Problems of searching and navigation have been studied long before the advent of microcomputers. A well-known strategy for traversing a maze is the *right-hand rule* or *wall-following* strategy: by following the wall while keeping one hand (either the left or the right one) always in touch with the wall, it is guaranteed to find a way out of a maze. The right-hand rule requires that the starting point has a connection to the outer walls of the maze. Though this strategy fails in cases where the walls are not connected, the right-hand rule is often applied for traversing obstacles or barriers in position-based routing and online navigation. If the target position is known, navigation algorithms can measure the progress towards the target and define a point where the obstacle boundary can be left. The robot motion-planning algorithms described in Section 2.5.1 are based on this idea. The following algorithms solve the traversal problem for arbitrary mazes.

One of the early theoretical works on maze searching, which uses a graph representation of a maze, is Trémaux's algorithm [Luc92]. It was already known in the 19^{th} century as a method for traversing mazes, following a depth-first search strategy. The algorithm traverses each edge exactly twice (in opposite directions). When arriving at a node it first uses edges that were not traversed in the same direction and finally leaves the node by the initial edge.

The problem of escaping from a maze is solved by the Pledge algorithm. This algorithm requires touch sensing and a compass so that the cumulative turn can be recorded. The algorithm can be described by the following instructions. Choose an initial direction and move in this direction whenever it is possible. When hitting an obstacle, follow the walls using the right-hand rule and count the degrees of rotation when changing the direction. Once the rotation counter is equal to zero, continue walking in the initial direction. The Pledge algorithm is able to find an exit, even if it is started inside the maze, whereas the wall-following method using the right-hand rule has to be started at the outer walls of the maze. The algorithm was named for John Pledge, who developed this navigation method at an age of 12. A description of the algorithm and a proof of correctness is presented by Abelson and diSessa [Ad82]. Further maze traversal algorithms and variants are described by Rao et al. [RKSI93] as well as by Lumelsky and Stepanov [Lum87].

2.4.4 Complexity of Labyrinth Problems

The requirements of Trémaux's algorithm and the Pledge algorithm are beyond the capabilities of finite automata; Trémaux's algorithm uses marks on the edges of the graph and the Pledge algorithm needs a rotation counter. This raises the question whether finite automata are able to search mazes.

Budach [Bud75] gave a negative answer to that question: there is no finite automaton that can search all mazes (see [Bra03] for a proof). A maze is modeled as a two-dimensional checkerboard with obstacles (cf. Figure 4.2). An automaton in a maze has the advantage of having a compass. With the information about the orientation, a maze

is easier to search than a planar graph [BK78]. Coy [Coy77] considered other computational models and showed that a pushdown automaton is not able to search a maze, but a linear space-bounded Turing machine can perform this task. Blum and Kozen [BK78] showed that even a logspace-bounded algorithm can solve this problem. They also showed that a two pebble automaton can search all mazes where the two pebbles simulate a counter. Furthermore, two cooperating automata can perform this task together. More about maze searching abilities of automata can be found in [Hem77].

In this thesis the algorithms are studied in a less restricted model than finite automatons: Counting is allowed as well as parallelism: in a network, an unbounded number of cooperative automata is available. However, communication between the parallel processes is not possible and this is one of the major problems in parallel online routing.

2.5 Robot Motion-Planning Algorithms

Motion planning is an important research area in robotics and has been studied under several aspects for various kinds of robots, including autonomous vehicles as well as manipulator robots, which are used in industrial assembly (see [HA92, Lat93, LaV06]). Many planning algorithms work offline, i.e. they require the environment to be known a priori. In the following we will concentrate on online planning algorithms that work in unknown environments. These algorithms solve navigation problems, which are related to the position-based routing problem. The online planning algorithms described in the next subsection use essentially the same idea as the greedy forwarding strategies with right-hand recovery in position-based routing. They perform greedy steps towards the target and circumvent obstacles by using the right-hand rule.

2.5.1 Online Planning in Continuous Environments

In the early 1980s Lumelsky and Stepanov began their work on robot motion planning algorithms. They developed two basic algorithms, Bug1 and Bug2, for navigation in an unknown environment using only touch sensing and analyzed their performance [LS84, Lum87, LS87, LS86]. Bug1 (see Algorithm 1) moves towards the target and follows boundaries of all obstacles it encounters. It leaves the obstacle boundaries at a local minimum, i.e. at the point that is nearest to the target. This is illustrated in Figure 2.8.

Bug2 uses a guide line connecting source and target. It follows the guide line until the target is reached or an obstacle is hit. Then it performs a partial traversal of the obstacles using the right-hand rule until the guide line is reached again. Figure 2.9 shows a trajectory of this algorithm.

Lumelsky and Stepanov express the performance of their algorithms in terms of the perimeters of all obstacles and the shortest path. The path length of Bug1 is at most $D + 1.5p$, where D is the Euclidean distance between source and target and p the sum of perimeters of all obstacles. This matches the lower bound for this class of algorithms

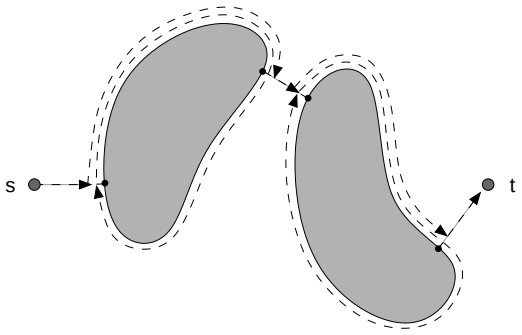


Figure 2.8: *Trajectory for Bug1*

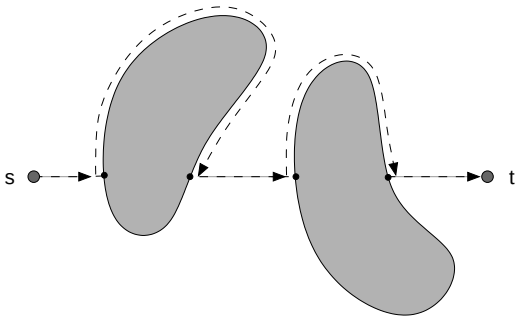


Figure 2.9: *Trajectory for Bug2*

Algorithm 1 Bug1

```
1: repeat
2:     Move towards the target along a straight line.
3:     if a barrier is hit then
4:         Start a complete right-hand traversal around the barrier
           and remember the point p that is nearest to the target.
5:         Take the shortest path to p (along the obstacle boundary).
6:     end if
7: until target is reached
```

that use touch sensing [Lum87]. Bug2 is more efficient in the case of convex obstacles, which are traversed only partially. However, in scenarios with arbitrary obstacles it can happen that parts of the obstacles are traversed several times. A comparison of the path length performance of Bug1, Bug2, and the maze searching algorithms is given in [Lum91].

A modification of Bug1 was proposed by Lucas [Luc88]. This variant uses the guide line of Bug2, performs a complete obstacle traversal and proceeds on the guideline at the nearest intersection point (i.e. the point where the guide line intersects the obstacle boundary). We use this algorithm as part of the alternating algorithm, which is a strategy to reduce time and traffic overhead at the same time. A detailed description follows in Section 4.4.1.

The work of Lumelsky and Stepanov has inspired several Bug-like strategies. Sankaranarayanan and Vidyasagar [SV90a, SV90b] proposed two variants of these algorithms that attempt to avoid a complete traversal of the obstacles: Alg1 and Alg2 leave the obstacle if a local minimum is reached (like Bug1) or the guide line is reached (like Bug2). When an already visited point is reached while traversing the obstacle, then the direction is turned and the obstacle is traversed in the opposite direction using the left-hand rule. The path length of these algorithms is at most $D + 2p$.

Other strategies were designed, e.g. to reduce the average path length [NHN03,

NNH04], work under position and orientation errors [NY98] or incorporate information from vision and range sensing [LS90, KRR98, Shi00]. An overview of these strategies is also given in [LaV06] and [RKSI93].

2.5.2 Motion Planning in Grid-based Environments

The underlying model for the planning algorithms described in the last section is continuos representation of the environment. There are other models that are based on the representation by a grid or a quadtree. Offline planning on a grid can be done by breadth-first search (BFS) or A* algorithms (see [HA92] for some examples). These algorithms are not suitable for online planning for a single robot, because they require access to all leaves of the search tree. A single robot cannot jump to other parts of the environment in order to continue the search in a more promising direction [LaV06]. With a team of robots, the navigation tasks can be performed at different places, but without global communication, the coordination is costly.

Stentz's D* algorithm [Ste94] is a dynamic variant of an A* algorithm that produces optimal trajectories for a single robot in a partially known environment. It maintains estimates of the cost for reaching the goal and updates these cost values during execution. Therefore, it also works in an initially unknown environment. The trajectories are optimal based on the information the robot had during execution. However, in an unknown environment a robot with touch sensing and restricted vision has to collect this information. This would lead to a traversal of obstacles as it is done by the Bug-like strategies.

The disadvantage of grid-based map representations is the large memory requirement. Therefore, a subdivision of the plane has been used in path-planning to overcome this problem. The quadtree is a popular choice for constructing efficient data structures. However, a standard quadtree-based approach for path planning cannot guarantee that Euclidean shortest paths are found. Chen et al. [CSU97] proposed a data structure called the *framed quadtree* to overcome this problem. The framed quadtree contains for each leaf additional information about the quadratic region (called *frame*) it represents. This information consists of the border nodes of a frame (border nodes correspond to the frame nodes in this work). A path through the environment enters and leaves obstacle-free frames and consists of straight-line segments that connect these entry points. To determine the shortest path, a modified BFS (called *wavefront*) is started that has to keep track of all these entry points because traversing a frame on a straight line between different entry and exit points causes different costs. This algorithm computes the Euclidean shortest path in time $\mathcal{O}(P) + \mathcal{O}(T_n)$, where P is the overall number of border cells (of obstacle-free leaves) in the quadtree and T_n the time for updating the pointers to neighboring frames. In our notation, P is $\mathcal{O}(p \log d)$, where p is the perimeter of all obstacles and d the length of the shortest path.

The algorithm by Chen et al. can be extended to find conditional shortest paths (i.e. shortest obstacle-free paths based on known information) in an unknown environment where the path-planning has to be updated when the information about new obstacles

arrives. It has been used in connection with the Stentz's D* algorithm [YSSB98] and was also extended to the 3-dimensional case [CSU95].

Wavefront propagation algorithms were also proposed for other path-planning problems (cf. [LaV06]). Hershberger and Suri [HS99] developed an optimal algorithm for Euclidean shortest paths in the plane among polygonal obstacles. This algorithm also uses a quadtree-style subdivision and requires $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n \log n)$ space where n is the number of obstacle vertices.

The JITE algorithm (see Section 4.6) also uses a subdivision of the environment that is similar to the quadtree-based approaches for path planning. In path planning, the environment is subdivided until the squares are obstacle-free (or contain only obstacles), whereas the JITE algorithm allows frames that are intersected by barriers as long as they do not cause a too long detour. This is crucial for its efficiency.

For searching on the subdivision, the JITE algorithm also uses a breadth-first strategy. Although this idea to use a BFS on a quadtree is similar, the planning algorithm by Chen et al. is a sequential offline algorithm. The JITE algorithm performs a parallel search, and this search is performed *online*, i.e. the information of one leaf of the BFS tree is not available at other leaves. This has to be coordinated in order to prevent a second traffic-consuming exploration of an area that has already been examined. The challenges both algorithms have to face, are quite different: The planning algorithm has to map Euclidean shortest path on the rectilinear frames of the quadtree while the BFS is executed, whereas the JITE algorithm has to construct the subdivision during the execution of the BFS and to coordinate the parallel branches of the BFS tree.

3

Position-based Routing using a Cell Structure

Position-based routing protocols for wireless networks use positioning capabilities (e.g. GPS) to make forwarding decisions. These protocols do not rely on the maintenance of routes, nor do they require a route discovery. Instead, packets are forwarded in the geographic direction of the packet's destination. Therefore, these protocols are scalable and suitable in dynamic networks like wireless networks. The main prerequisite is that a node in the network knows its geographical position and the position of the destination. If each node broadcasts its own position to the neighbors at regular intervals, then all nodes know their neighbors' positions and are able to choose the next node that minimizes the distance to the destination. This is the basis for *greedy forwarding* (see Section 2.1). If greedy forwarding fails in case of a local minimum, a *recovery strategy* is executed that guides the message around this region, e.g. by applying the right-hand rule. Such strategies often require a planarization of the network topology to prevent routing loops. For planarization, several subgraph constructions have been proposed as described in Section 2.1.3. Unfortunately, the planarization cancels out the long edges of the original network topology, so that the routing progress is reduced just in the case when the effective greedy strategy fails.

The Cell-based Geographic Forwarding Protocol (CGFP), which is presented in Section 3.3, avoids this disadvantage of planarization. It is based on a geographical clustering and provides each node with a map of its environment that contains the necessary information for greedy forwarding and recovery. This map is constructed by subdividing the plane into cells which are classified as *link cells* or *barrier cells*. Figure 3.1 illustrates this idea. Link cells indicate communication regions, whereas barrier cells represent void regions where no communication is possible. The map can be appended to beacon messages, which are necessary for announcing position information. This way, a node obtains 2-hop information with only a constant data overhead of the beacon messages. Moreover, the cell structure provides an implicit planarization that is required for loop-free paths in recovery mode (i.e. if a barrier is traversed). The Cell-based Geographic Forwarding Protocol provides a distributed construction of the cell structure using only local information. It constructs routing paths based on the cell

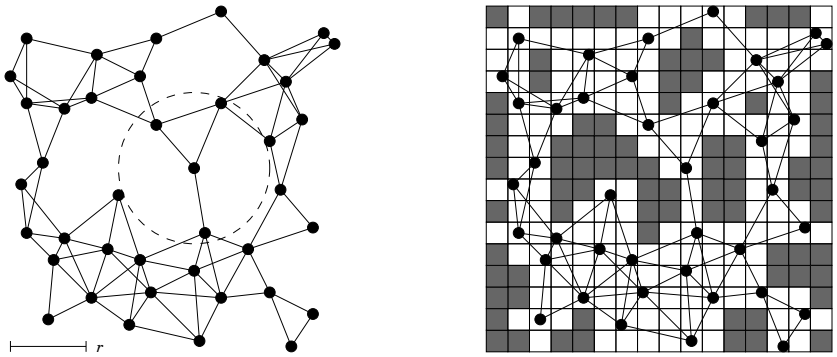


Figure 3.1: *Unit disk graph with transmission radius r (left) and grid subdivision (right) of a node set.*

structure by a greedy forwarding strategy with right-hand recovery.

In Section 3.4 we will see that paths on the cell structure are equivalent to paths in the original network. Thus, position-based routing in wireless networks (under the unit disk graph assumption) is essentially the same as routing on the cell structure, which can be regarded as a two-dimensional mesh with barriers. In Chapter 4 this model is used to study the impact of parallelism on the efficiency of position-based routing strategies.

3.1 The Network Model

A wireless network can be modeled as a graph $G = (V, E)$ with $|V| = n$, where the nodes V are the participants of the network and the edges E are the communication links. The nodes are placed in the Euclidean plane and have a fixed transmission radius r. We assume uniform transmission ranges: there is a link between the nodes u and v if and only if $||u - v|| \leq r$. For a normalized transmission radius ($r = 1$) the resulting graph is called *unit disk graph* (UDG). Figure 3.1 (left) shows an example.

3.2 Proactive Versus Reactive Information Dissemination

For position-based routing, a node has to know its own position and the position of the target. We also assume that the positions of the neighbors are known. If this requirement should be fulfilled, then the information has to be collected beforehand. This leads to the question whether this information exchange is necessary.

In a continuous time model, a proactive exchange of position information is not needed. There are approaches for position-based forwarding without beacon messages (see Section 2.1.4). Instead of providing neighboring nodes with position information as basis for their forwarding decisions, the decision about the next hop is based on the response time of the neighboring nodes: nodes with the most suitable locations an-

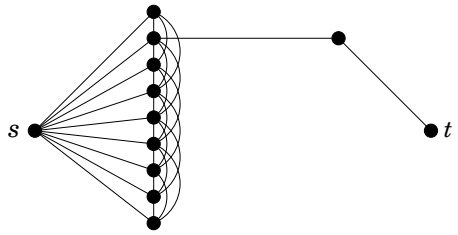


Figure 3.2: *Worst-case example for reactive routing*

swer earlier than others. However, these protocols require a continuous time space for distinguishing between response times that represent the positions of the neighboring nodes.

The assumption of a discrete time model is justified by the fact that the response time of a node in a wireless network is limited, and therefore in practice a continuous medium access protocol relying on arbitrarily small time differences does not work. Moreover, in the worst case, a completely reactive communication protocol needs as much messages as used by proactive beaconing.

Theorem 3.1 *Every reactive position-based communication protocol needs $\Omega(n)$ message transmissions or $\omega(1)$ rounds for delivering a single message in the worst case even if the hop distance between source and target is constant.*

Proof: The proof of this theorem follows the ideas presented in [BGI92] using the situation depicted in Figure 3.2. The source s can reach $n-3$ nodes directly. But only one of them is on the shortest path of length 3 from s to the destination node t. In the first round s can inform all nodes with one message except two nodes. Then, only one of them can forward it on the right path toward the destination. Which one is not known to the network and cannot be determined by the use of the destination coordinates. Therefore a deterministic algorithm can be forced to try $n-3$ attempts until succeeding by a fooling argument.

For a probabilistic algorithm we choose the first node randomly from the neighbors of s. If the attempt fails, we retry and choose one of the remaining neighbors. Hence the expected number of attempts to succeed is $n/2-1$. Thus the necessary messages to deliver the packet is at least $n/2+1$. ■

This theorem shows that a minimum proactivity is helpful. Thus, we concentrate on communication protocols that use a proactive exchange of position data. Then the question is how the position information can be exploited. The easiest way to disseminate this information is to include position coordinates in beacon messages that are broadcasted at regular intervals. This way every node knows the nodes in its neighborhood and their positions. This position data can also be forwarded to the next neighbors, so that each node also knows the position of the 2-hop neighbors. This requires to add a list of neighbors to the beacon messages and this list can be of arbitrary length. Distributing information over multiple hops and maintaining this information can help

to detect local minima in static scenarios, but in the case of dynamics and mobility this leads to the problem of out-dated information. Also the scalability is no longer ensured. Thus, the proactive information dissemination would contradict the idea of position-based routing.

The Cell-based Geographic Forwarding Protocol exploits the 2-hop information, but uses beacon messages with only constant data overhead. Instead of distributing a list of the neighbors' positions, each node broadcasts a map of its environment to its neighbors. This map contains information about a constant number of cells, which are classified as link cells or barrier cells.

In the next subsection we show how to establish a geographic clustering by distributing 2-hop information. The information about a node's environment is only broadcasted to the direct neighbor and not distributed any further.

3.3 The Cell-based Geographic Forwarding Protocol

The Cell-based Geographic Forwarding Protocol (CGFP) consists of two parts: the distributed construction of the cell structure and the forwarding algorithm.

The first part of the protocol provides the basis for the forwarding. It establishes a grid subdivision of the plane containing the information about the neighborhood of the nodes: each node subdivides the area covered by its transmission radius into *cells*. By exchanging beacon messages with position coordinates, each node knows the position of its direct neighbors. Then, the cells between a node and a direct neighbor can be classified as *link cells*, the other cells are *barrier cells* or remain *unclassified*. With this classification a node has some kind of map containing the local information of the area covered by its transmission radius. This map is called *view*. The information from a view is appended to the beacon messages, so that other nodes can extend their views. This way a node obtains the information about its 2-hop neighborhood with only a constant overhead in the size of the beacon messages. This part of the protocol is described in Section 3.3.1.

The second part of the protocol contains the forwarding of messages. If a node receives a message (and if it is not the destination), it has to select one neighboring node and send the message to it. This selection is based on the position of the destination and the local information in the view, which is collected in the first part. The view is used for a virtual forwarding of the packet from cell to cell using greedy forwarding with recovery. The link cells in the view indicate regions that can be used for forwarding, barrier cells indicate regions where no node can take over a message for further forwarding. If a virtual path leaves the own cells or reaches the end of the map (unclassified cells) then the message is physically forwarded to another node. This part of the protocol is described in Section 3.3.2. A detailed description can be found in the specification [BR06].

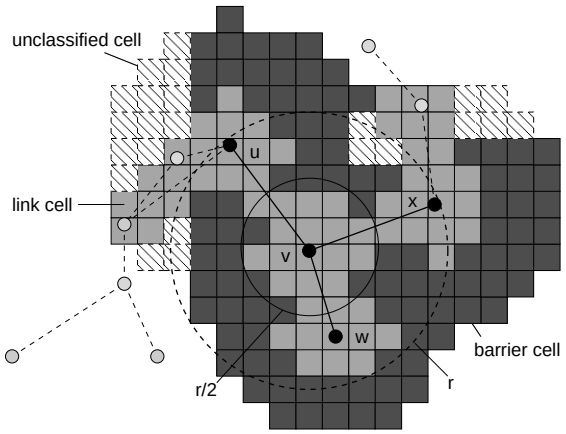


Figure 3.3: *Classification of cells*

3.3.1 Establishing the Cell Structure

Each node creates a map of its environment by subdividing the area covered by its transmission radius into cells. The cells in this *cell structure* are classified in order to determine those regions that can be used for routing and those who cannot. The side length ℓ of a cell is determined by the transmission radius r. We will see in Section 3.4 how ℓ has to be chosen so that a correct classification of link cells is guaranteed.

Classification of cells

A cell is classified by a node v as

- **link cell** if it lies completely within half of the transmission radius of v of any neighboring node (*half radius rule*) or if it is intersected by an edge (*edge intersection rule*).
- **barrier cell** if all of the following conditions are fulfilled: it lies completely within the transmission radius r of v, it is closer to v than to any other neighboring node, and it is not a link cell.
- **unclassified** if it is not covered by the transmission radius, or if it is not a link cell, but a neighboring node is closer to it than v.

Views

For the cell classification two views are used: **View 1** contains the set of cells within the radius r and their classification. This classification is based on the positions of the direct (1-hop) neighbors. The information of this view is given to the neighboring nodes in the beacon messages. **View 2** is the enlarged view of a node. It contains the cells classified by the node (View 1) and by the classifications of the neighbors that add their

Beacon	
Field	Description
ID	ID/network address of the sender
pos	geographic position of the sender
view	array containing type and owner of cells from View 1 of the sender

Figure 3.4: *Contents of a beacon message*

classification to the beacon messages. View 2 is the basis for routing (forwarding). It is not included in the beacon and not sent to other nodes.

View 1 is a grid, whose side length is $2\lfloor r/\ell \rfloor + 1$ so that the the complete transmission radius is covered. Unit for the dimension is the number of cells. View 2 should also cover the transmission range of potential neighbors, therefore a side length of $4\lfloor r/\ell \rfloor + 1$ is chosen. Each cell in a view has a *type* and an *owner*. The owner is either the node itself or a neighbor. The direct neighbors are stored in a list where their geographical positions can be looked up.

The views are initialized as follows. In the beginning, all cells are unclassified. Then, the cells which are *completely* within the half radius of the node are set to link cells. Cells between half-radius and transmission radius are set to barrier cells if no other neighbor is known who is closer to them. After this step, the cell status is updated when beacon messages arrive.

Updating the Cell Classification

The views have to be updated if neighbors appear, move or disappear, or if the node that hosts the view moves. As the own movement affects the link cells within the half radius and link cells intersected by links to the neighbors, View 1 has to be rebuilt from the position information stored in the list of neighbors. The movement or appearance of neighbors are recognized by beacon messages, their disappearance can be handled by timeouts. If a node receives a beacon message from a neighbor, it updates the views as follows:

1. If the neighbor is already known and if he has moved, then delete the old information in View 1 and View 2 (reset the cells owned by the neighbor).
2. Classify the link cells by applying the half radius rule and the edge intersection rule in View 1.
3. Copy information about cells owned by the neighbor to View 2.
4. Copy classified cells from the beacon message into View 2.

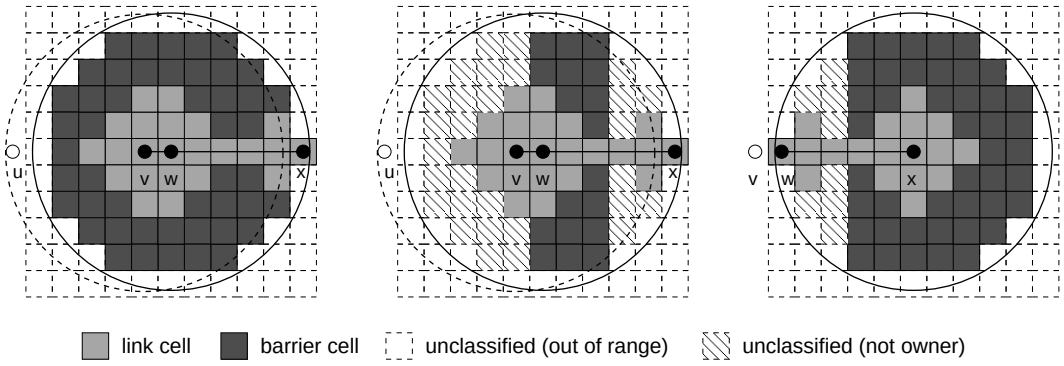


Figure 3.5: *View 1 of node w with an incorrect classification (left), with a correct classification (middle) and View 1 of node x (right)*

The update of the views from a beacon message (Steps 2-4) is described by UPDATE-CELLS (Algorithm 2). A beacon message includes the geographic position and View 1 of the sender (see Figure 3.4).

The problem of classifying a barrier cell correctly is that the information about this cell is only reliable if no other node could classify this cell differently. Assume four nodes u, v, w and x are placed on a line as shown in Figure 3.5. Node w knows only v to the west. Node u is beyond its range, and it can only be reached by node v. But w does not know this. It marks the cells west of node v as barrier cells (in View 1, beyond v's half radius) and sends this information eastward to node x. So in View 2 of node x there are only barrier cells to the west of v. Then, node x would not attempt to physically forward the packet in that direction, though there is a route. This problem can even occur if the two nodes in the middle (v and w) are in the same cell.

Thus, a node has to be sure that a cell is a barrier cell, i.e. that there must be other node that could classify this cell differently. This means for View 1: a cell is classified as barrier cell if no other node is closer to this cell. Otherwise they are marked as unclassified.

The correct classification for the example is shown in Figure 3.5. Node w does not classify the cells to the west of v as barrier cells. So beyond the link cells around v there are only unclassified cells. This information is passed to x and used to create View 2 (see Figure 3.6). If x performs the cell-based forwarding by constructing a path westward in View 2, it reaches the unclassified cells and then it has to hand over the packet to node w. Node w has the information about u in its View 2 and will forward the message further in this direction. If u and v are deleted from the picture, then there are no other nodes to the west of w and the cells beyond w's half radius can be classified as barrier cells.

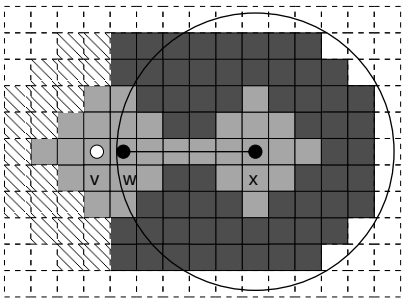


Figure 3.6: *View 2 of node x (containing the information of View 1 of node v)*

As a consequence of the problem of barrier classification, a node v leaves a cell C unclassified if *all* of the following conditions are true:

- C is inside the transmission radius r of node v.
- C lies outside the half-radius of node v.
- C lies outside the half-radius of any neighbor (i.e. it is not a link cell).
- The distance from C to a neighbor is smaller than the distance to v.

If a new neighbor appears, all barrier cells that are closer to the neighbor than the node itself, have to be set to unclassified. If a neighbor disappears, then each cell that was owned by this neighbor has to be checked whether it is a link cell of another node. If this is not the case than it can be classified as barrier only if there is no other neighbor closer than the node itself. Otherwise, it has to be unclassified.

View 2 is constructed by taking the information from the beacon messages. When a node v receives a beacon message from a neighbor u, then it adopts the classification from View 1 of the neighbor, but with the following restrictions:

- The own cells in View 2 are not overwritten.
- The own cells from the beacon message (owner=v) are ignored.
- Unclassified cells from the beacon message are ignored, except the cell in View 2 is owned by u.
- Barrier cells from the beacon message cannot overwrite link cells in View 2.

If a classification of a cell is adopted, then u becomes the new owner in View 2 of v.

The update of the cell classification is described by UPDATECELLS (Algorithm 2). It uses the following functions:

- UPDATEVIEW1(cell, type, owner) assigns type and owner to the cell in View 1. A cell of the same type is only overwritten if the position of the new owner is closer to the cell than the old owner.
- UPDATEVIEW2 works analog to UPDATEVIEW1. It forbids that unclassified cells overwrite cells of another type by another owner in View 2.

Algorithm 2 UPDATECELLS(beacon)

```
 1: neighbor = beacon.ID
 2: neighbor_cell = local cell containing beacon.pos
 3: for each cell(x,y) in view1 do
 4:     if cell.owner ≠ me and GETDISTANCE(me.pos,cell) < r then
 5:         d = GETDISTANCE(beacon.pos,cell)          // distance from cell to neighbor
 6:         if d < r/2 then
 7:             UPDATEVIEW1([x,y], type:link, owner:neighbor)
 8:         else                                      // if there are barrier cells that are closer to the neighbor:
 9:             if cell.type = barrier and (cell.owner = none or cell.owner = neighbor)
                  and d < GETDISTANCE(me.pos,cell) then
10:                 UPDATEVIEW1([x,y], type:unclassified, owner:neighbor)
11:             end if
12:         end if
13:     end if
14: end for
15: SETLINKCELLS([0,0], neighbor_cell, neighbor)
16: for each cell(x,y) in view1 do                    // copy updated cells into View 2:
17:     if cell.owner = neighbor and (cell.type≠barrier or view2[x,y].type≠link) then
18:         UPDATEVIEW2([x,y], cell.type, cell.owner)
19:     end if
20: end for
21: for each bcell(x,y) in beacon.view do             // adopt cell classification from the beacon
22:     vcell = view2[x+neighbor_cell.x, y+neighbor_cell.y]
23:     if vcell.owner≠me and bcell.owner≠me
            and (bcell.type≠unclassified or vcell.owner = neighbor)
            and (bcell.type≠barrier or vcell.type≠link) then
24:         UPDATEVIEW2([x,y], bcell.type, owner:neighbor)
25:     end if
26: end for
```

- SETLINKCELLS(cell1, cell2, neighbor) sets all cells that are intersected by the line segment between the own position (me.pos) and the neighbor's position to link cells and ensures that these cells are orthogonally connected. The owner of each cell is either this node (me) or the neighbor, whichever is closer to the cell.
- GETDISTANCE(position, cell) returns the maximum distance from the geographical position p to any of the corners of the cell c.

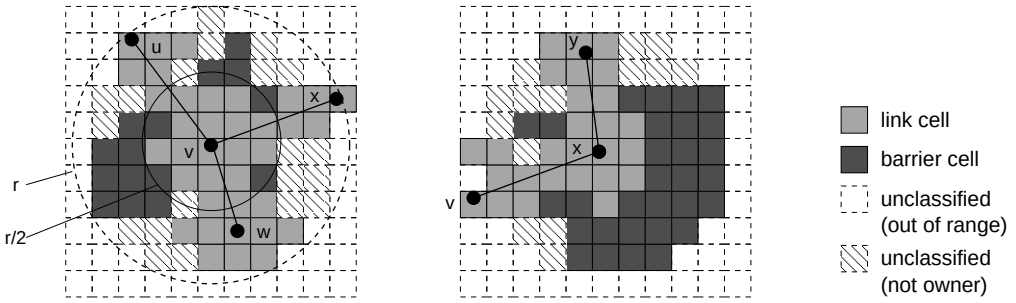


Figure 3.7: *Node v (left) and node x (right) and the classification of cells based only on the positions of the direct neighbors (View 1)*

An Example for the Cell Classification

Figure 3.7 shows View 1 of nodes v and x. The link cells within the half radius are owned by v and x. Further link cells are determined by the position of the direct neighbors. The remaining cells are either barriers or they remain unclassified if a neighbor is closer to them. Figure 3.8 shows the enlarged view (View 2) of node v after it has received beacon messages of all its neighbors. In this view the information from v's View 1 and the views of the neighbors are merged by UPDATECELLS. The missing information in v's View 1, e.g. the unclassified cells east of v (Figure 3.7 left) are provided by the neighbors x and w.

The complete picture of the network with further nodes is shown in Figure 3.3. From node v's point of view the links outside the transmission radius are unknown.

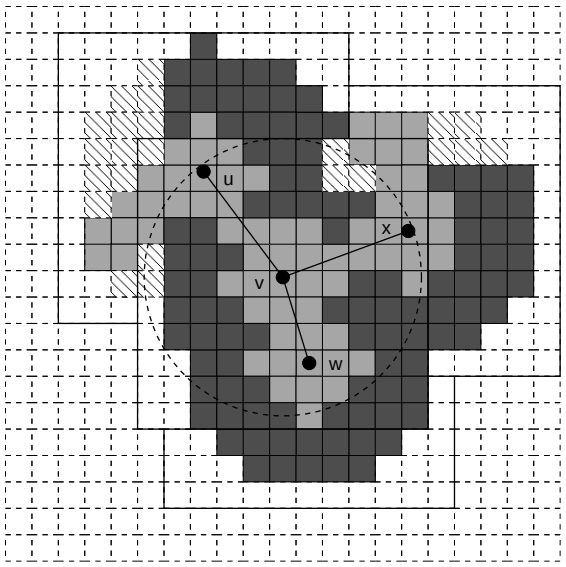


Figure 3.8: *The enlarged view of node v (View 2)*

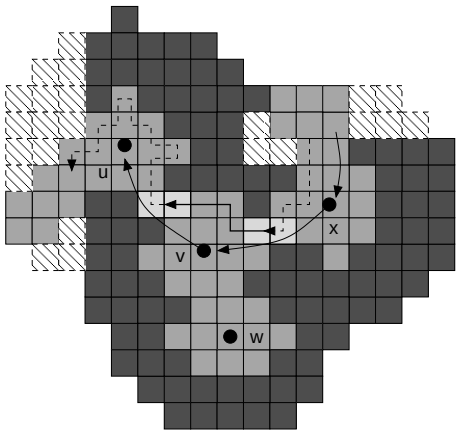


Figure 3.9: *Construction of a right-hand traversal path in View 2*

3.3.2 Cell-based Routing

If a node receives a message and if it is not the destination, then it has to forward this message to another node. If one of the neighbors is the destination node, then it can simply forward the message to him (see PROCESSPACKET, Algorithm 3). Otherwise it has to select a neighbor that has the most promising geographical position. This decision is made by means of the cell structure: the received packet contains the information about the current cell (where it virtually resides). From this cell the node constructs a cell path in View 2 by applying a greedy and recovery strategy until the own cells are left or an unclassified cell is reached. Then the owner of this cell is the neighbor to whom the packet is forwarded.

In greedy mode a packet is virtually forwarded from cell to cell by choosing always the orthogonally adjacent cell that is nearer to the target. If the next cell towards the target is a barrier cell, then no progress is possible and the recovery mode is used. In recovery mode the barrier is traversed using the right-hand rule. The greedy forwarding is resumed if the *progress condition* is fulfilled, i.e. if the packet is nearer to the target than the position that was recorded when starting the recovery.

Routing on the cell structure is performed in the local view of a node. Figure 3.9 shows a right-hand traversal path in View 2 that is constructed after a node v received a packet from its neighbor x. When the own cells are left, the packet is physically forwarded to another neighbor u that has to continue the cell path construction. So the packet has to contain all necessary information about the current state. The contents of a packet is listed in Figure 3.10.

The routing on the cell structure is described by FORWARDPACKET (Algorithm 4). FORWARDPACKET performs a greedy forwarding with right-hand recovery on the cells in View 2 as long as the cells are owned by this node. If the owner changes, then the packet is physically forwarded to the owner of the next cell. The next cell is determined by

Packet	
Field	Description
source	ID/network address of the sender
dest	destination address
dest_pos	geographic position of the destination
cell_pos	current global cell position of the packet
dir	current direction of the packet
mode	forwarding mode, either greedy or recovery
rec_cell	global cell position when entering recovery mode
rec_dir	direction when recovery was started

Figure 3.10: *Contents of a packet*

GETNEXTGREEDYCELL or GETNEXTRECOVERYCELL depending on the current mode of the packet (lines 5–9). If a cell is traversed a second time in the same direction, then the target is not reachable and the packet can be dropped (line 11).

GETNEXTGREEDYCELL (Algorithm 5) determines the orthogonally adjacent cell that is nearest to the target. If this next cell is a barrier, then the packet is switched to recovery mode and the current direction is turned to the left so that the barrier is to the right-hand side. The current direction and the current position are stored in the packet, so that it can be checked later whether the progress condition is fulfilled or if the packet returned to this position.

GETNEXTRECOVERYCELL (Algorithm 6) determines the next cell in recovery mode, i.e. when the border of a barrier is traversed. When the recovery mode starts, the barrier is to the right. So the first step in this function is a right turn, and if no barrier is there, then this is the next cell. Otherwise, a left turn is repeated until a non-barrier cell is found. The recovery mode ends if the progress condition is fulfilled.

If only the first turn to the right is carried out and the next cell is unclassified, then the packet is physically forwarded to a neighbor. In the neighbor's view this cell could be a link cell or a barrier cell. The *justTurnedRight* flag is used to adjust the direction so that the recovery mode can continue correctly in the neighbor's view (line 16 in FORWARDPACKET).

To avoid symmetry problems, unclassified cells that are owned by the sender of the packet can be regarded as barrier cells.

Algorithm 3 PROCESSPACKET(packet)

```
1: if packet.dest = me.ID then
2:     hand over packet to application layer
3: else if packet.dest in list of neighbors then
4:     SEND(packet, packet.dest)
5: else
6:     FORWARDPACKET(packet)
7: end if
```

Algorithm 4 FORWARDPACKET(packet)

```
1: current_cell = GETLOCALCELLPOS(packet.cell_pos)
2: dest_cell = POS2CELL(packet.dest_pos)
3: stop = false
4: while not stop do
5:     if packet.mode = greedy then
6:         next_cell = GETNEXTGREEDYCELL(packet, current_cell)
7:     else
8:         next_cell = GETNEXTRECOVERYCELL(packet, current_cell)
9:     end if
       // if a cell is traversed twice in the same direction, then the destination is unreachable:
10:    if next_cell = packet.rec_cell and
             packet.rec_dir = (next_cell – current_cell) then
11:        drop packet
12:    end if
13:    if next_cell.owner ≠ me or next_cell is unclassified in view2 then
14:        stop = true                                   // stop if the own cells are left
15:        if packet.mode = recovery and justTurnedRight then
16:            TURNLEFT(packet.dir)
17:        end if
18:    else
19:        current_cell = next_cell                      // go one step ahead
20:    end if
21: end while
22: packet.cell_pos = GETGLOBALCELLPOS(current_cell)
23: SEND(packet, view2(current_cell).owner)          // send pkt. to owner of this cell
```

Algorithm 5 GETNEXTGREEDYCELL(packet, current_cell)

```
1: source_cell = POS2CELL(packet.source_pos)
2: dest_cell = POS2CELL(packet.dest_pos)
3: packet.dir = GETDIRECTION(source_cell, current_cell, dest_cell)
4: next_cell = current_cell + packet.dir        // next_cell = pos. of the cell one step ahead
5: if view2[next_cell.x,next_cell.y].type = barrier then
6:     packet.mode = recovery
7:     packet.dir = TURNLEFT(packet.dir)        // turn left in front of a barrier
8:     packet.rec_dir = packet.dir              // direction for starting recovery
9:     packet.rec_cell = GetGlobalCellPos(current_cell)   // store current cell
10:    return current_cell
11: end if
12: return next_cell
```

Algorithm 6 GETNEXTRECOVERYCELL(packet, current_cell)

```
1: packet.dir = TURNRIGHT(packet.dir)           // turn right, then the barrier is ahead
2: justTurnedRight = true;                      // remains true if there is no further turn
3: next_cell = current_cell + packet.dir
4: while view2[next_cell.x,next_cell.y].type = barrier do
5:     justTurnedRight = false;
6:     packet.dir = TURNLEFT(packet.dir)        // barrier is to the right after first turn left
7:     next_cell = current_cell + packet.dir
8: end while
9: if GETDISTANCE(dest_pos,next_cell) < GETDISTANCE(dest_pos,packet.rec_cell)
   then
10:    packet.mode=greedy                       // resume greedy if progress condition fulfilled
11: end if
12: return next_cell
```

The forwarding algorithms use the following functions:

- SEND(packet, address) sends the packet physically to the node with the specified address.
- GETDIRECTION(cell1, cell2) returns the direction of cell1 to the orthogonally neighboring cell that is nearest to cell2.
- GETDISTANCE(position, cell) returns the maximum distance from the global position to any of the corners of the cell.
- TURNLEFT(direction) or TURNRIGHT(direction) return the direction after a 90-degree turn to the left (counter-clockwise) or to the right (clockwise).
- GETGLOBALCELLPOS(localpos) and GETLOCALCELLPOS(globalpos) convert local to global cell coordinates and vice versa.
- POS2CELL(position) returns the global cell coordinates of the given position.

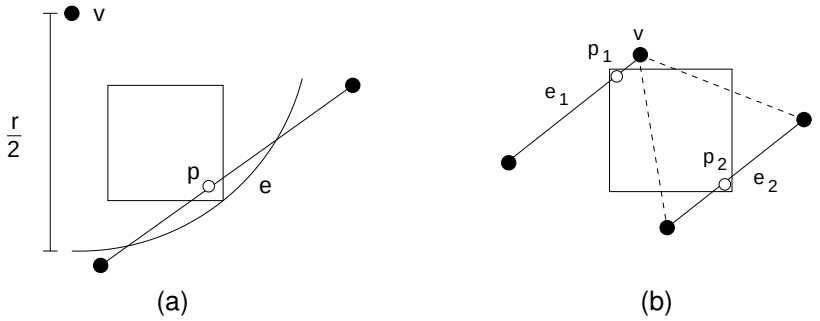


Figure 3.11: *Illustrations for the proof of Lemma 3.1*

3.4 Equivalence of Network and Cell Structure

When using the cell structure for routing instead of the original network topology, virtual paths are created that can be turned into paths in the network. We will see that a path in the cell structure and a path in the network are equivalent up to a constant factor with respect to the number of hops, or the number of cells, respectively. We define such cell-based paths formally as follows.

Definition 3.1 *A* cell path $(C_1, \ldots, C_m)$ *consists of link cells, such that* C_i *and* C_{i+1} *are orthogonally neighboring cells.*

First we have to ensure that the owners of adjacent cells can reach each other. The owner of a link cell is one of the nodes whose position has led to the classification of cells. These nodes are the implicants of a cell.

Definition 3.2 *We call an edge the* implicant *of a link cell if it intersects with the link cell. Similarly we call a node the implicant of a link cell if all points in the link cell have maximum distance* $r/2$ *to the node.*

We show that paths in the network and cell routes are essentially equivalent. The equivalence is based on the *connectivity property* of a link cell.

Connectivity Property: A link cell C fulfills the connectivity property if there is a direct connection (i.e. the distance is less or equal to r) between all the implicants for C. In case of an edge being the implicant this is required for at least one of the incident nodes.

Lemma 3.1 *If* $\ell \leq \sqrt{\frac{3}{8}}\, r$ *then all link cells fulfill the connectivity property.*

Proof: In the case of two nodes being the implicants of a link cell, this follows by the triangle inequality, since all points of the link cell have maximum distance $r/2$ to the nodes. In the case of a node and an edge as implicants, there is a point in the link cell, which has at most distance $r/2$ to one of the nodes of the edge and the connectivity property follows.

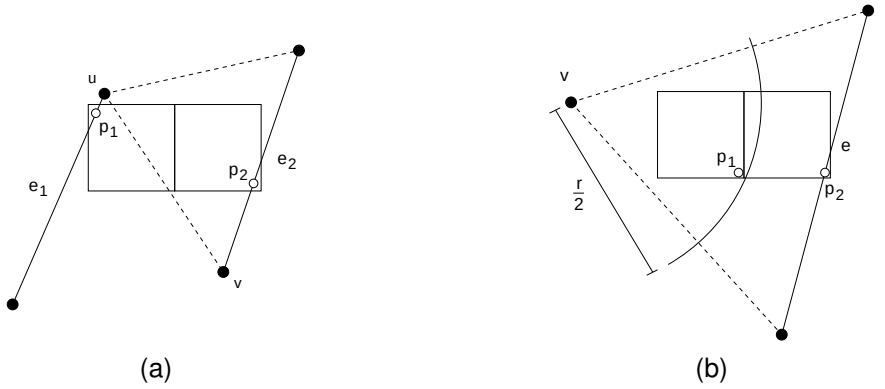


Figure 3.12: *Illustrations for the proof of Theorem 3.2*

If two edges are implicants for the link cell then the question is: how can the distance between the closest nodes incident to different edges be maximized? In a worst case a node v incident to one edge e_1 is equally distant to both nodes of the other edge e_2, i.e. these three end points form an isosceles triangle. Since there exist two points p_1 on e_1 and p_2 on e_2 with $||p_1 - p_2|| \leq \sqrt{2}\,\ell$ (causing e_1 and e_2 being implicants of the same cell), the distance between v and one point on e_2 (the mid point) is at most $||p_1 - p_2|| \leq \sqrt{2}\,\ell$. Then, the maximum distance between v and $u \in e_2$, which should not exceed r, is $\sqrt{2\ell^2 + r^2/4}$. This leads to the bound $\ell \leq \sqrt{3/8}\,r$. ■

Theorem 3.2 *If $\ell \leq \frac{1}{2}(\sqrt{3}-1)\,r$, then paths in the network and cell routes in the cell structure are equivalent up to a constant factor:*
1.) Every path P of the network can be replaced with a cell path P' of length at most $2\lceil \frac{r}{\ell} \rceil |P| + 1$ containing all nodes of the path P, where $|P|$ is the number of nodes on the path.
2.) For every cell path P' there is a path P of length at most $2\,|P'|$, where $|P'|$ is the number of cells on the path.

Proof: 1.) The path $P = (u_1, \ldots, u_n)$ is substituted as follows. For each edge $e_i := (u_i, u_{i+1})$ we add the link cells that are intersected by e_i. If we measure the length of each edge e_i by using the Manhattan distance between u_i and u_{i+1} based on the cell size, we obtain $2\lceil \frac{r}{\ell} \rceil |P| + 1$ cells for the whole path.

2.) For two orthogonally neighboring cells on a cell path the connectivity of the implicants has to be guaranteed. The following considerations refer to the connectivity of the implicant nodes and edges:

a) Implicant edges: consider two edges e_1 and e_2, each of them implicant of one of the two link cells (see Figure 3.12 a). Then, there are two points p_1 on e_1 and p_2 on e_2 with $||p_1 - p_2|| \leq \sqrt{5}\ell$. The maximum distance between p_1 and the nearer incident node of e_2 is reached if both nodes of e_2 are equally distant to p_1. If $r \geq \sqrt{r^2/4 + 5\ell^2} \Rightarrow \ell \leq \sqrt{3/20}\,r$, then p_1 is connected to at least one node of e_2.

The case of p_2 and e_1 is symmetric. This bound guarantees connectivity between implicant edges, i.e. if $\ell \leq \sqrt{3/20}\, r \approx 0.387\, r$ then $\exists u \in e_1, v \in e_2 : ||u - v|| \leq r$.

b) Implicant edge and implicant node: consider a node v that is implicant of the cell C and an edge e that is implicant of the neighboring cell (see Figure 3.12 b). The distance between v and any point p_1 in C is at most $r/2$. Thus, there is a point p_2 on e with $||p_1 - p_2|| \leq \ell$ and $||v - p_2|| \leq \frac{r}{2} + \ell$. The connection of v and one of the nodes incident to e is guaranteed if $r^2 \geq (\frac{r}{2} + \ell)^2 + (\frac{r}{2})^2 \Rightarrow \ell \leq \frac{1}{2}(\sqrt{3} - 1)r \approx 0.366\, r$.

c) Implicant nodes: since the distance of each point inside the cell to the implicant node is at most $r/2$, the two implicant nodes of neighboring cells are always connected.

The transitions between the cells on the cell path P' are transformed as follows: We start with $P = \{u_1\}$ where u_1 is one node in the first link cell of P'. A node u is responsible for a cell C if u is inside C or implicant of C or incident to an edge that is implicant of C. We substitute a transition from a cell C_i to C_{i+1} as follows: if C_i is a link cell containing the last node of P, then we add a node to P that is responsible for C_{i+1}. If C_i is a link cell for which the last node u_i of P is an implicant, then u cannot always reach a node u_{i+1} that is responsible for C_{i+1}. From the considerations above and the connectivity property we know that there is another node v responsible for C_i that is in reach of u_i and u_{i+1}. We add v and u_{i+1} to P. So we add at most two nodes to P for one cell of P'. ■

3.4.1 Distributed Route Construction

As the views of the nodes in the network contain only the cell classification of their local environment, a cell path that runs across several views has to be constructed distributedly. Therefore, we have to ensure that all transitions between cells can be performed locally or by a physical forwarding to a neighbor that is responsible for the target cell. The next lemma shows that a transition from an own link cell to another link cell in the own view is also a valid transition in the view of a neighbor.

Lemma 3.2 *Given two nodes u and v. For each pair of adjacent cells (C_1, C_2) in View 2 of node u holds:*

a) If C_1 and C_2 are link cells and C_1 is owned by u and C_2 is owned by v, then C_1 and C_2 are also link cells in View 2 of v with the same owners.

b) If C_1 is a link cell and owned by u and C_2 is a barrier cell, then C_2 is not a link cell in View 2 of v.

Proof: a) As the case $u{=}v$ is obvious, we concentrate on the case $u{\neq}v$. If the connectivity property is fulfilled, then there is always a connection between the implicant nodes

and the nearest node incident to an implicant edge of two adjacent cells. As the owner of a cell is an implicant and has the smallest distance to the cell among all implicants, the owners of adjacent cells know each other and all other implicants. Therefore, it is not possible that C_1 (and also C_2) has different owners in View 2 of u and View 2 of v.
b) Assume that C_2 is a link cell in View2 of node v. Then it is adjacent to C_1 for which u is an implicant. Thus, u must have a connection to the implicant of C_2 and cannot classify it as barrier. ■

Lemma 3.2 shows that a transition between two link cells can be handled locally by a node if it is the owner of both cells is its View 2. If the owner of the target cell is a neighbor, then it is guaranteed that both start and target cell have the same classification in View 2 of the neighbor. A cell path can be constructed locally in the following way:

- If the next cell is a link cell: if the owner is me, then go to the next cell. If the owner is a neighbor, then forward the message physically to the neighbor.
- If the next cell is a barrier cell, then stop.
- If the next cell is an unclassified cell, then forward the message physically to the owner of the unclassified cell.

These rules are considered by the forwarding functions of the Cell-based Geographic Forwarding Protocol.

3.5 Conclusion and Outlook

We have seen that routing on the cell structure and geographic routing in a wireless network are equivalent under the unit disk graph assumption. The model of a faulty mesh network, which is presented in Chapter 4 is a further abstraction of the cell structure. Thus, the online routing algorithms that work on a faulty mesh network can also be applied in wireless networks if position information is available. Furthermore, the performance measures for routing in faulty mesh networks can be adopted, since the length of paths in the cell structure is equivalent to the length of paths in the original network.

Non-uniform Transmission Ranges and Obstacles

The equivalence of the wireless network and the cell structure was shown under the unit disk graph assumption. This assumption is violated if the transmission ranges are not uniform or in the presence of obstacles that obstruct radio signal propagation. In this model, barrier cells represent void regions in a wireless network, but this notion can be also extended to obstacles. The concept of barrier cells is compatible to this notion as long as the connectivity property of link cells is satisfied. In Figure 3.13 an example is given where this property is violated by obstacles. In such a case the connectivity graph is not planar anymore and position-based traversal algorithms or face routing algorithms fail. A greedy strategy would run into a local minimum (node w in

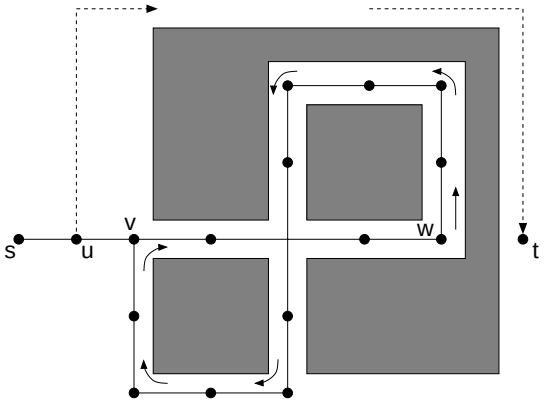


Figure 3.13: *Position-based traversal strategies fail in the presence of obstacles.*

the figure), then a right-hand traversal gets into a loop. Approaches that handle such cases are described in Section 2.1. However, the detection of crossing links requires communication along paths in the network, and these paths may become as large as the diameter in the worst case.

Outlook

In a dynamic network where nodes join and leave the network or move around, message delivery cannot be guaranteed in every situation. Even flooding algorithms may fail if the network is temporarily partitioned. For an up-to-date cell classification, it is more important to obtain the newest information than to get a correct assignment of the owners of the cells. The use of timestamps for the cell classification sent in beacon messages would help to prevent that up-to-date information is overwritten.

Another aspect is the performance of the protocol, which can be increased by the following extensions. Both greedy forwarding and recovery are based on the cell structure and the physical forwarding takes place when the own cells are left. In greedy mode, the owner of the next cell is not always the node that provides the most forwarding progress. Thus, the next hop in greedy mode could be determined by selecting the neighbor providing the most progress within the transmission radius (MFR), while the cell structure is used only in recovery mode. The performance in recovery mode could be increased if the protocol uses more information about the neighborhood, which is given in View 2: if the physical forwarding is performed as late as possible, intermediate hops can be reduced and possible dead ends can be detected beforehand. The drawback of this idea is that a node would have to rely on the information from the neighbors only and this can lead to errors especially in dynamic scenarios. In scenarios with a high node density the collisions between transmitted data packets become more likely. Then, the geographic clustering that is implicitly given by the cell structure could be used to define a channel access scheme.

The protocol is implemented and evaluated in a simulation environment by Inés Bebea González as part of her Master's thesis [Beb06].

4

Online Routing in Faulty Mesh Networks

Routing in mesh networks has been intensively studied in the field of distributed and parallel computing. In these networks nodes may fail or may be unavailable and this failure can often only be detected by its neighbors. A straight-forward approach is to regularly test the neighbors of each node, to collect this information and distribute a table of all failed and working nodes throughout the network. We investigate scenarios where this knowledge is not available before the route discovery starts.

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 maze-like structures are possible. For a route discovery the source knows the target position, but the rest of the network is unknown. This problem is essentially the same as the position-based routing problem in wireless networks or the online navigation problem (see Sections 2.1 and 2.4.1). In a situation where the network is unknown, 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 to circumvent barriers when searching a path to the destination increases the time.

In this chapter we will consider different techniques to solve the route discovery problem in faulty mesh networks and analyze how much time they need and how much traffic they generate. We start our considerations with a description of the model and the performance measures in the following sections. Section 4.3 shows lower bounds for time and traffic. In Section 4.4 we review two basic routing strategies: a single-path algorithm, which is slow but traffic-efficient, and a multi-path or flooding algorithm, which is asymptotically time-optimal but causes a large traffic overhead. Section 4.5 shows how a combination of these two basic strategies can result in an algorithm that is efficient in both time and traffic. Finally, Section 4.6 presents the Just-in-Time-Exploration (JITE) algorithm that solves the route discovery problem asymptotically as fast as flooding, but causes much less traffic.

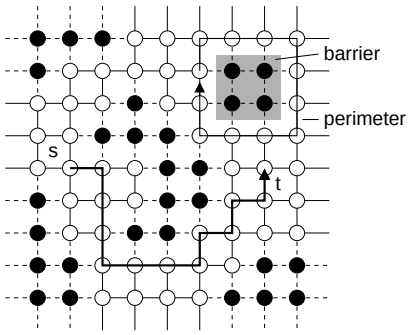


Figure 4.1: *Mesh network with faulty nodes (black), routing path and right-hand traversal path*

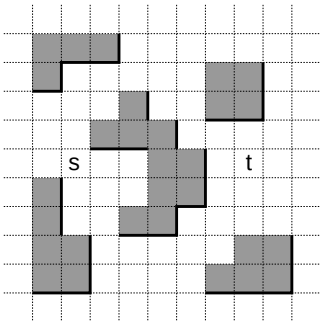


Figure 4.2: *Faulty meshes are equivalent to mazes or grid polygons with obstacles.*

4.1 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 source node and target node in the network. We will see that the major impact on the efficiency of the routing algorithm is not influenced 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.

In the following the source node is denoted with s and the target node with t. The length of the shortest barrier-free path connecting s and t is denoted with d.

4.1.1 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 that 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 $p := \sum_{i \in \mathbb{N}} p(B_i)$. In other words, 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.

4.2 Comparative Measures

Online algorithms are usually analyzed under comparative measures, because the worst-case analysis of online problems often does not lead to meaningful results. If we consider the straight-line distance D between source and target, then the worst case scenario is a labyrinth of barriers, leaving only one path of length $\mathcal{O}(D^2)$ to the target. So any algorithm has to traverse the labyrinth. However, if the scenario contains less barriers, the situation is quite different.

In the following we define comparative measures for time and traffic that take the difficulty of a scenario into account. The difficulty is expressed by the shortest path and the perimeters of the barriers.

4.2.1 The Competitive Time Ratio

Time is the number of steps needed by the algorithm to deliver a message. This 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 [BE98].

Definition 4.1 *An algorithm A has a* competitive ratio *of c if*

$$\forall x \in \mathcal{I} : C_A(x) \leq c \cdot C_{\text{opt}}(x)$$

where $\mathcal{I}$ is the set of all instances of the problem, $C_A(x)$ the cost of algorithm A on input x and $C_{\text{opt}}(x)$ 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 4.2 *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.*

4.2.2 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. Thus, 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 [KP00]:

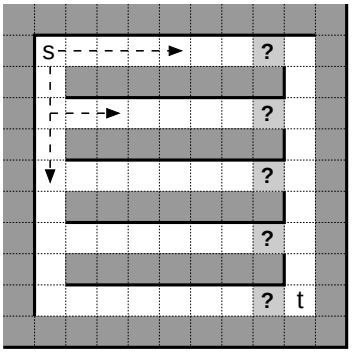


Figure 4.3: *The lower bound scenario of Theorem 4.1. The adversary controls the "exits" which are marked with "?".*

Definition 4.3 *An algorithm A 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 inherent in 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 an optimal offline algorithm can be found: $\max_{x \in \mathcal{I}_{\{d,p\}}} \min_{B \in \mathcal{B}} C_B(x) = d$. This requires the modification of the comparative ratio in [KP00] in order to obtain a fair measure. Thus, we use the online lower bound for traffic to define the *comparative traffic ratio*.

Definition 4.4 *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.*

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

Definition 4.5 *The* combined comparative ratio *is the maximum of the competitive time ratio and the comparative traffic ratio: $\mathcal{R}_c := \max\{\mathcal{R}_t, \mathcal{R}_{Tr}\}$*

4.3 Lower Bounds

An offline algorithm, which has global knowledge, can determine a path to the target in time d and with 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. Clearly,

the target cannot be reached with less than d time steps, but we are also interested in the traffic that is needed.

The following theorem shows the online lower bound for traffic. A similar proof is given by Blum et al. [BRS97] 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 [Lum87].

Theorem 4.1 *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 4.3). 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 perimeters of all obstacles and the enclosure, therefore $p = k \cdot 2l + \mathcal{O}(1)$. 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 4.1 *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.*

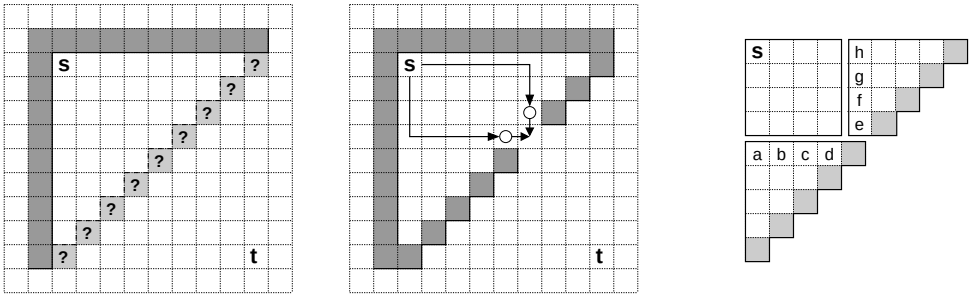
The following table summarizes the lower bounds for time and for traffic.

	Time	Traffic
Online lower bound, single-path	$\Omega(d + p)$	$\Omega(d + p)$
Online lower bound, multi-path	$\Omega(d)$	$\Omega(d + p)$
Best offline solution	d	d

Note that time and traffic are considered independently. This leads to the question whether there is a trade-off between time and traffic. A linear trade-off is shown in the next section.

4.3.1 A Trade-off between Time and Traffic

The considerations about the lower bounds for single-path strategies and multi-path strategies do not answer the question how much traffic has to be invested to increase the time. The JITE algorithm presented in Section 4.6 shows a poly-logarithmic upper bound on the combined comparative ratio. It is an open problem whether this bound is tight or whether there is a poly-logarithmic trade-off between time and traffic. It



(a) Triangle of barriers. The diagonal is controlled by the adversary.

(b) The exit in the diagonal can be reached from two outposts

(c) The triangle splitted into three parts.

Figure 4.4: *Illustrations for the proof of Theorem 4.2*

is obvious that saving a constant number of messages increases the time by at least a constant number of steps. But it is not obvious that there is a linear trade-off, i.e. that reducing traffic by a factor c_1 increases the time by at least a factor of c_2 for two constants c_1 and c_2.

In order to show a linear trade-off, we have to consider a scenario where the algorithm has the choice between several paths leading to the target that can be obstructed by the adversary. This basic idea is used in the proof of Theorem 4.1, but the scenario in that proof is a worst case for traffic. Therefore it is not suitable here, because there is no chance to save traffic—not even at the expense of time. Here, we consider a small scenario with a total perimeter size of $\Theta(d)$ where the minimum time and the number of messages can be counted exactly. Then we construct a series of these scenarios, where in each block either time or traffic can be saved. This leads to the linear trade-off.

Theorem 4.2 *There is at least a linear trade-off between time and traffic, i.e.* $\boldsymbol{Tr} + c \cdot \boldsymbol{time} = \Omega(d)$ *for a constant* c*.*

Proof: Consider a game between algorithm and adversary. Source and target are placed in a square of size 9×9 as shown in Figure 4.4(a). The diagonal is controlled by the adversary, i.e. the diagonal may consist of barriers, but at least one exit has to remain open. The optimal time for reaching the target is 16 steps. Achieving optimal time is only possible if certain points in front of the diagonal are reached within 7 steps, such that from these points the exit in the diagonal can be found in the next time step. In the following these points are called *outposts*. They are depicted with small circles in the figures. It is not necessary to reach all of the points in front of the diagonal, because an exit in the diagonal can be reached from two neighboring points as shown in Figure 4.4(b).

The goal of the algorithm is to find a path to the target. The goal of the adversary is to hinder the algorithm by placing a barrier on the diagonal each time the algorithm tries to traverse it. However, one exit has to remain open. Thus, if the algorithm should reach

the target in optimal time, it has to establish enough outposts so that the whole diagonal is covered, i.e. that for any exit in the diagonal there is always an adjacent outpost, from which the diagonal can be crossed. Once the exit is reached, the adversary constructs a new triangle that has to be traversed by the algorithm (see Figure 4.5).

First, we determine the minimum traffic any algorithm has to spend for constructing enough outposts to cover the diagonal. For that we have to consider all possible trees that are rooted at the source and whose leaves are the outposts. We simplify this case study by splitting the triangle into two sub-triangles and a square as shown in Figure 4.7. Then the total traffic is given by the traffic inside the sub-triangles and traffic needed to traverse the square (including the exits).

The sub-triangles can be entered from one or two entrances. If there are more than two entrances, at least two entrances are adjacent. Two adjacent entrances are equivalent to one entrance and a branch inside the sub-triangle, i.e. the traffic remains the same because saving one message inside the triangle is compensated by an additional entrance. So we neglect adjacent entrances and concentrate on the cases shown in Figure 4.6. This figure shows for each single entrance and each combination of non-adjacent entrances a traffic-optimal message tree. The minimum traffic when using a particular entrance (or two entrances) is summarized in the following table.

Sub-triangle		Entrance(s)	Traffic	Case (Fig. 4.6)
Lower triangle	1 entrance	a	7	L1
		b, c, d	6	L2, L3, L4
	2 entrances	a+c	5	L5
		a+d, b+d	4	L6, L7
Right triangle	1 entrance	e	6	R4
		f, g, h	5	R1, R2, R3
	2 entrances	e+g, e+h, f+h	4	R5, R6, R7

Now, we consider the square containing the starting point, which is shown in Figure 4.7. Reaching both the right upper corner and the left lower corner of the 4×4 square from the starting point takes at least 6 messages. Each exit requires an additional message. The following table shows the combinations of using one or two exits from the square and the resulting traffic for the square (including the exits).

Case	# Entrances		Minimum traffic			Total traffic	Illustrations
	L	R	L	R	S	(L+R+S)	(Fig. 4.6, 4.7)
1	1	1	6	5	8	19	S1
2	1	2	6	4	9	19	S2
3 a	2	1	4	6 (e)	9	19	S2; R4
3 b	2	1	4	5 (f)	10	19	S3, S4; R3
4	2	2	4	5	12	21	S5

L = lower triangle, R = right triangle, S = square

In case 1 there are 8 messages needed for the square, in case 2 an additional message is

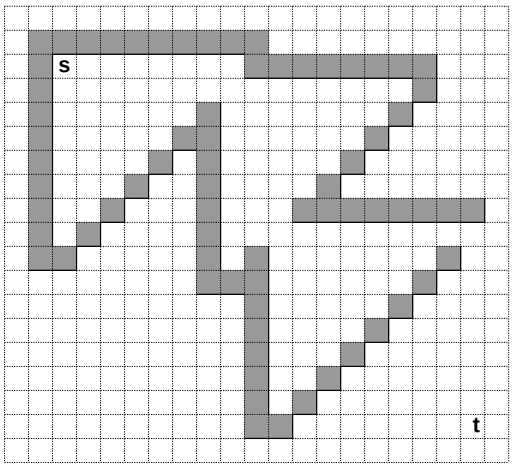


Figure 4.5: *The complete scenario for the lower-bound (Theorem 4.2)*

needed because of the additional exit. The total traffic for these two cases is simply the sum of the minimum traffic for the separate parts. For case 3 we have to distinguish between (a) using exit e (6 messages in the right triangle) and (b) using one of the other exits (5 messages in the right triangle). In case 3 b there are 10 message required inside the square because two exits can be reached with 8 messages, but the three exits in this case are not adjacent and it requires one additional message to reach the third exit and one more message for the exit itself (see S3 and S4 in Figure 4.6). In case 4 there are 12 message needed for the square because from the tree leading to b and g there must be a path to the exits e and d (see S5 in Figure 4.6). This path requires two additional messages, since the exits on the right side as well as on the lower side are not adjacent. This leads to more than 19 messages in total. Thus, establishing outposts that cover the whole diagonal requires at least 19 messages.

If each exit in the diagonal should be reached in optimal time, the lower left corner and the upper right corner of the triangle have to be reached directly, i.e. the exits a and h have to be used. These are the cases L1 and R1 (for one entrance), and L5, L7, R6 and R7 (for two entrances). Two exits from the square (leading to L1 and R1) require 8 messages and result in a total traffic of 20. Three exits require 11 messages because there is an additional non-adjacent exit that is two steps from the direct path between starting point and exit a or exit h. Adding the minimum traffic for one sub-triangle with one entrance and another one with two entrances also results in 20 messages. Finally, four exits require at least 12 messages. As the minimum traffic for a sub-triangle is 4, this case requires also 20 messages in total.

Thus, reaching the target in optimal time is only possible with 20 messages. The minimum traffic of 19 can only be reached if one of the exits a or h is avoided, but then a detour of two steps is added to either the path to the lower left corner or the upper right corner.

Now, consider a cascade of k triangles as shown in Figure 4.5. We know that each

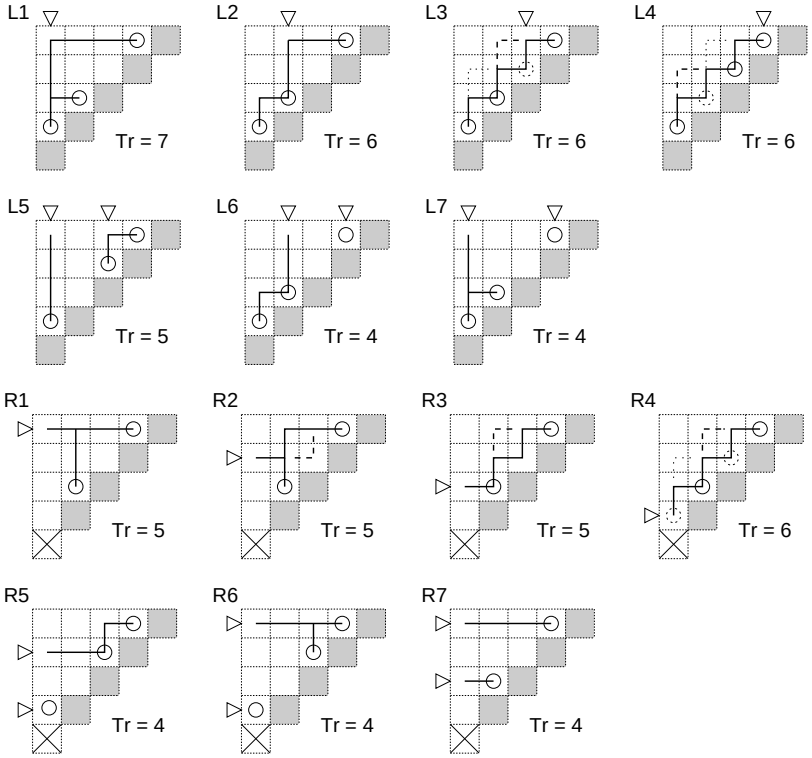


Figure 4.6: *Optimal traffic bounds for the lower sub-triangle (L1–L7) and for the right triangle (R1–R7) and different entrances (Theorem 4.2)*

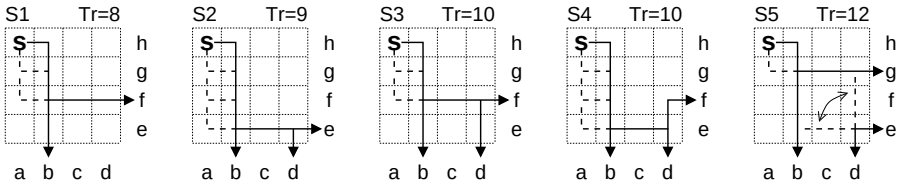


Figure 4.7: *Traffic-optimal paths in the sub-square (Theorem 4.2)*

triangle can be traversed with 20 messages without delay in at least 8 time steps or with 19 messages in at least 10 time steps. Assume m of the triangles are traversed without delay ($m \in \{0, ..., k\}$). Then, **time** $\geq 8m + 10(k - m) + 8$ and **Tr** $\geq 20m + 19(k - m) + 8$. The shortest path has length $d = 9k + 8$ (9 steps to reach the next triangle and finally 8 steps to reach the target).

$$\begin{aligned} \Rightarrow \quad & \textbf{Tr} \geq -\tfrac{1}{2}\,\textbf{time} + 29k + 16 \\ & = -\tfrac{1}{2}\,\textbf{time} + \tfrac{29}{9}(d - 8) + 16 \\ \Rightarrow \quad & \textbf{Tr} + \tfrac{1}{2}\,\textbf{time} = \Omega(d) \end{aligned}$$

■

4.4 Basic Strategies

In this section we review two basic strategies: Lucas' algorithm, a traffic-efficient single-path strategy, and expanding ring search, a time-efficient flooding algorithm. A comparison shows the basic problems of both approaches. We also describe a variant of expanding ring search, that is later used for the JITE algorithm.

4.4.1 Lucas' Algorithm

Lucas' algorithm [Luc88] is a simple single-path strategy that follows the straight line connecting source and target and traverses all barriers that intersect this guideline (see Algorithm 7). It was proposed as a modification of an online path-planning algorithm of Lumelsky and Stepanov [LS86] (see Section 2.5.1).

Algorithm 7 Lucas' algorithm

```
1: repeat
2:     Follow the straight line connecting source and target.
3:     if a barrier is hit then
4:         Start a complete right-hand traversal around the barrier
           and remember all points where the straight line is crossed.
5:         Go to the crossing point that is nearest to the target.
6:     end if
7: until target is reached
```

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 algorithm matches the asymptotic lower bound for single-path online algorithms and also the asymptotical lower bound for traffic.

4.4.2 Expanding Ring Search

A straightforward multi-path strategy is *Expanding Ring Search* [JM96, PBD03], 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.

The following comparison between expanding ring search with Lucas' algorithm shows the advantages and disadvantages of these strategies. Lucas' algorithm performs well if there are few barriers, but the sequential traversal of a maze takes too much time. Expanding ring search works efficiently in a maze, but in open space it needs more messages than necessary. Thus, both strategies fail in optimizing time and traffic at the same time.

Exp. Ring Search	$\mathcal{R}_t$	$\mathcal{R}_{Tr}$
General	$\frac{\mathcal{O}(d)}{d}$	$\frac{\mathcal{O}(d^2)}{d+p}$
Open space ($p < d$)	$\mathcal{O}(1)$	$\mathcal{O}(d)$
Maze ($p = d^2$)	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Lucas' algorithm	$\mathcal{R}_t$	$\mathcal{R}_{Tr}$
General	$\frac{\mathcal{O}(d+p)}{d}$	$\frac{\mathcal{O}(d+p)}{d+p}$
Open space ($p < d$)	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Maze ($p = d^2$)	$\mathcal{O}(d)$	$\mathcal{O}(1)$

Both expanding ring search and Lucas' algorithm have a combined comparative ratio of $\mathcal{O}(d)$.

4.4.3 Continuous Ring Search

We modify the expanding ring search 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 the JITE algorithm.

Lemma 4.1 *Continuous ring search with a slow-down of $\sigma > 1$ finds a path connecting s and t 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 $\sigma \cdot d$ where d is the length 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 $t/\sigma = \frac{\sigma+1}{\sigma-1} d$. ■

4.5 The Alternating Algorithm

The idea of the *alternating algorithm* [RS05a] is to combine a single-path strategy with flooding. Both strategies are applied alternatingly and with a time restriction. We will see that with this combination a sub-linear combined comparative ratio can be achieved. For the single-path strategy an algorithm is needed that guarantees to find the target in $\mathcal{O}(d+p)$ time steps, e.g. Lucas' algorithm. Both strategies are applied with a search depth restriction (time-to-live). We start with a small search depth and repeat the single-path strategy and flooding alternatingly with increased search depth until the target can be reached (see Algorithm 8).

Algorithm 8 Alternating Algorithm

```
1: i = 1
2: repeat
3:     δ = 2^i
4:     Start Lucas' algorithm with a maximum search depth of δ^(3/2)
5:     if target is not reached then
6:         Start Flooding with a maximum search depth of δ
7:     end if
8:     i = i + 1
9: until target is reached
```

At first sight, it seems that this algorithm combines the disadvantages of both flooding and the single-path strategy. But it turns out that it has a better combined competitive ratio.

Theorem 4.3 *Let d be the length of the shortest path connecting s and t. The alternating algorithm finds the target in time $\mathcal{O}(d^{3/2})$ and produces traffic $\mathcal{O}(\min\{d^2, d^{3/2}+p\})$.*

Proof: We distinguish between two cases:

Case 1: The target is reached while applying the single-path strategy (line 4). This strategy uses greedy paths (segments on the guide line) and traversal paths (along the border of a barrier). As the greedy rule is only applied after a progress towards the target was made when, the length of the greedy paths is bound by d. The overall length of the traversal paths is at most $\frac{3}{2}p$. The target is reached in $d+\frac{3}{2}p$ steps, i.e. within $\frac{2}{3}\log(d+\frac{3}{2}p)$ iterations (because $2^{3i/2} \leq d+\frac{3}{2}p$). The search depth for flooding in the previous iteration was $2^{2/3\cdot\log(d+\frac{3}{2}p)-1} = \frac{1}{2}(d+\frac{3}{2}p)^{2/3}$. In this case, the target could not be reached by flooding, but flooding yields optimal paths. Hence, d is larger than the search depth of flooding in the previous iteration: $\frac{1}{2}(d+\frac{3}{2}p)^{2/3} \leq d$. This implies

$d + \frac{3}{2}p \leq (2d)^{3/2}$. For time and traffic we obtain the following bounds:

$$\textbf{time} = d + \tfrac{3}{2}p + \sum_{i=1}^{\frac{2}{3}\log(d+\frac{3}{2}p)-1} \left(2^{3i/2} + 2^i\right) \leq 7\,d^{3/2}$$

$$\textbf{Tr} = d + \tfrac{3}{2}p + \sum_{i=1}^{\frac{2}{3}\log(d+\frac{3}{2}p)-1} \left(2^{3i/2} + 2^{2i}\right) \leq 2(d + \tfrac{3}{2}p)^{4/3}$$

Case 2: The target is reached while flooding (line 6). It takes $\log d$ iterations to approach the target through incremental flooding. In each iteration there is an additional delay caused by the single-path strategy (line 4). Thus, the time is bound by $\sum_{i=1}^{\log d} 2^{3i/2} + 2^i \leq 4\,d^{3/2}$. The target is only reached by flooding (with depth d) if the single-path strategy is forced to make a detour. But any barrier with a perimeter smaller than $d^{1/2}$ cannot prevent a message that traverses a barrier from returning to the guide line and then reaching the target. That means reaching the target by flooding occurs only if barriers of size $p \geq d^{1/2}$ are present. If there are smaller barriers ($p < d^{1/2}$) then the target can be reached by the single-path strategy and we can apply case 1. Otherwise, $p \geq d^{1/2}$ and the traffic is bound by $\sum_{i=1}^{\log d} 2^{3i/2} + 2^{2i} \leq 3\,d^2 \leq 3(d+p)^{3/2}$. ■

Corollary 4.2 *The alternating algorithm has a combined comparative ratio of* $\mathcal{O}(\sqrt{d})$.

Proof: Follows by the definition of the competitive time ratio and the comparative traffic ratio, and Theorem 4.3. ■

4.6 The JITE Algorithm

In this section the Just-In-Time Exploration (JITE) algorithm is presented. We start with an overview, which is followed by a detailed description of the most important part of the algorithm: the fast exploration.

4.6.1 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 4.9). If the target cannot be found inside this search area, new quadratic subnetworks in the environment are investigated. The search area is enlarged until the target is reached.

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, 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, 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. Note that this is not an exploration of the complete environment, as it is often described in the literature (cf. Section 2.4.2). The algorithm stops the further investigation of a region if there is definitely no possible path to the target.

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, so 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, so 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 finished in time.

4.6.2 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* of the frame nodes and the nodes on the perimeter of a barrier. As a frame can be partitioned by a barrier, we refer to a *partition* of a frame which is enclosed by frame nodes and perimeter nodes (see Figure 4.8). A *right-hand traversal path* in a partition of a frame is a path containing all frame nodes and perimeter nodes of the partition, 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 is accessible and that there are not too many border nodes.

Definition 4.6 *A partition of a $g \times g$ frame is* simple *if the partition contains a frame node v and if 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 $\gamma(t)$.

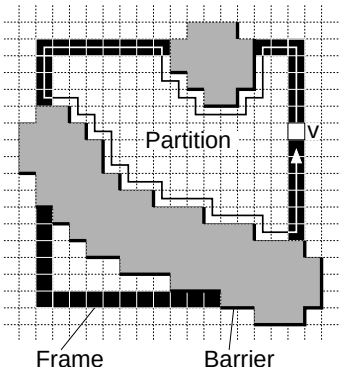


Figure 4.8: *Partition of a frame, defined by a right-hand traversal*

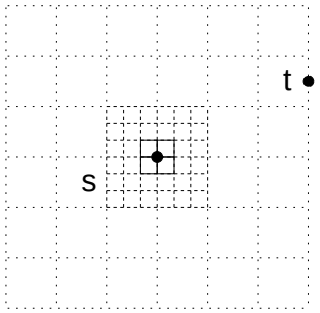


Figure 4.9: *Initial frames (solid) and extended search area*

Subdivisions

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, which 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 4.10): 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 would have 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.

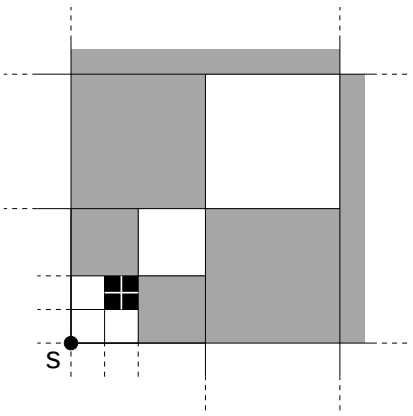


Figure 4.10: *Domino effect in a* 2×2*-subdivision. A subdivision of the black square triggers subdivisions in the shaded squares.*

The exploration of a frame basically consists of a "round-trip communication": 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 4.8). 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. 4.6) 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, so 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, this is a proof for too many barriers. But the reverse is not true. Only if the original message returns to the originating entry node and visits at most g/γ border nodes, 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, the partition is not simple and then there is no coordinator for the second round. Furthermore, if the traversal fails, 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 of the second round generates two "counting" messages. These messages travel clockwise and counter-clockwise on the frame nodes and perimeter nodes and carry a counter for visited frame nodes and border nodes. If the messages meet and if they counted together at most g/γ border nodes and at least $4(g - \frac{g}{\gamma})$ frame nodes, the partition is simple and the node where they meet becomes the coordinator for the third round. Otherwise there is no coordinator. In this way a simple partition can be checked within $2(1 + 1/\gamma)g$ steps.

3rd round, "stop": If the partition is simple, 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, 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.

With 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. Thus, 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. Therefore, $\Delta + 2(1 + 1/\gamma)g \geq 10(1 + 1/\gamma)\frac{g}{3} + \frac{\Delta}{3} \Rightarrow \Delta \geq 2$. Thus, 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 4.2, it follows that this time is bounded by $t \geq \frac{g}{3} - \frac{g}{\gamma} = g\frac{\gamma-3}{3\gamma}$. According to this, the frame size g' of the neighboring frame is chosen so 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 $\gamma(t)$.

4.6.3 Slow Search

The slow search uses a breadth-first search on the frames of the subdivision to propagates a slow proceeding shoreline through the network. The construction and subdivision of frames is done by the fast exploration algorithm, which is triggered by the shoreline. When the shoreline arrives, the frames are already constructed and the search for the target becomes as simple as the continuous ring search algorithm. A BFS is started on the frames of the subdivision in two waves. The first wave is slowed down by a constant factor. This factor is determined in Lemma 4.2. When the target is found and the source has been notified, a second wave is started on the generated paths in order to stop the exploration.

We can assume that the target lies on a frame of the subdivision, i.e. $||s - t||_\infty = 3^k$ for some $k \in \mathbb{N}$. 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.

In the following, time and traffic of the JITE algorithm are analyzed separately. The time bound can be achieved because the 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.

4.6.4 Time Analysis

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, neighboring frames that are not already explored are subdivided according to the subdivision rule (see Section 4.6.2). The following observation follows directly from this rule:

Observation 4.1 *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 side length of the new squares is increased by a factor of at most 3. Thus, there are no subdivisions of these new squares only because of the subdivision rule, unless they are intersected by barriers. The restriction on the size of neighboring squares is one prerequisite that each square can be explored just in time.

Lemma 4.2 *If the BFS is slowed down by a constant factor σ, 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 the shoreline initiates the sending of a "notify"-message to an entry node and this message "starves". Then there may not be 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 4.14), the distance to a node u on the diagonally neighboring frame B is at least $||u - v||_1 \geq \frac{g}{3} - \frac{g}{\gamma}$, as the shoreline comes from a neighboring subframe 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 so 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 so 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 4.3 *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' connecting s and t with length $|P'| \leq 2|P|$.*

Proof: We observe that the recursive subdivision produces a fine-grained grid in the proximity of barriers, so that all nodes adjacent to a barrier are part of the grid. If a part of P goes along a barrier, this part is also contained in the grid. If a part of P goes through "open space" and is not part of the grid, 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 4.4 *Let $|P|$ be 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 so that he is guaranteed not to become insolvent, 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 at 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 4.11 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 4.11 b). Thus, the traveler possibly has to pay twice for a square. If his financial strength allows to pay this extra cost, then it will suffice in either case.

Figure 4.12 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

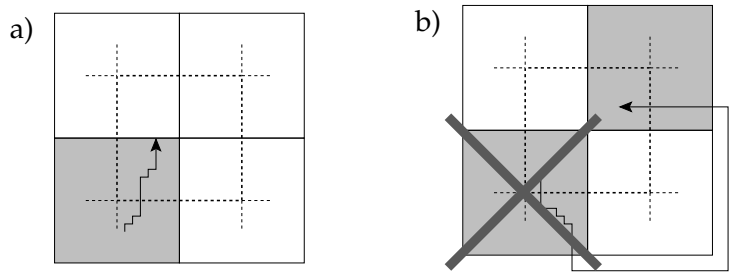


Figure 4.11: *Illustrations for the proof of Lemma 4.4. a) The transition between squares (dashed lines) takes place in a quadratic area, called the* scope *(shaded square). b) This transition cannot occur because the first mark is ignored when the scope changes.*

the horizontal or vertical borders of the scope. One can easily see that transitions lead either to case ① (→ A), where one square is already visited, or to case ② (→ 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 the traveler 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, the traveler needs three dollars when reaching this point. At the border of the scope he needs only two dollars, because on the way to the center he 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. Thus, $\gamma(t)$ 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.

Lemma 4.5 *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. Thus, 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, there are

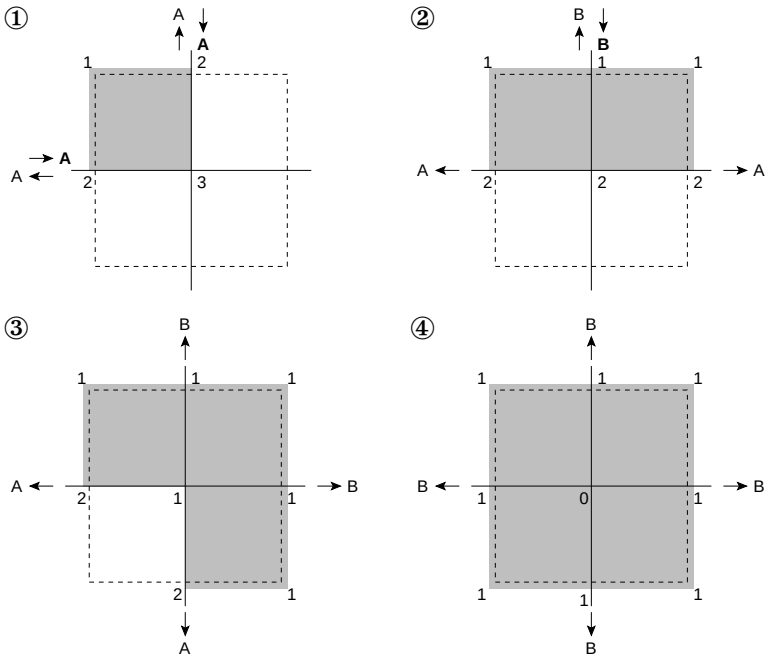


Figure 4.12: *Illustrations for the proof of Lemma 4.4. Marked squares, which have been visited before, are shaded. Transitions are denoted with arrows. The numbers show the costs at certain positions.*

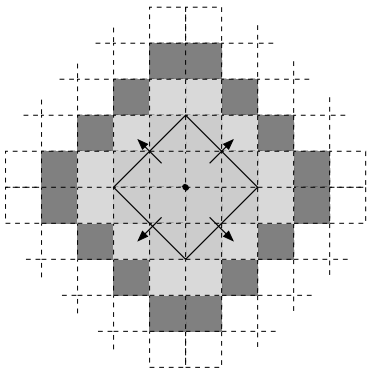


Figure 4.13: *Illustration for the proof of Lemma 4.5: the black diamond is the shoreline, in the shaded squares the exploration has started.*

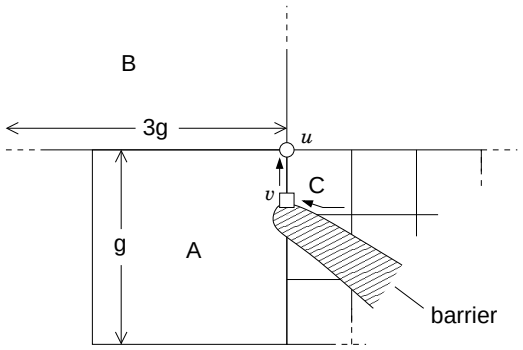


Figure 4.14: *The shoreline enters square A at node v. The exploration of square B must be finished, when the shoreline enters B.*

at most $2(r+2)(r+3) =: \varepsilon\, n_i$ squares where the exploration has started or is possibly finished (see Figure 4.13). 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. Thus, 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 4.4) and add the detour of $g/\gamma(t)$ per frame.

Theorem 4.4 *Let P be the shortest path with length $|P|$ connecting s and t. The JITE algorithm finds a path P' of length $\mathcal{O}(|P|)$ connecting s and t.*

Proof: The path P visits $\ell_i(P)$ frames of size 3^i. We know from Lemma 4.4 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:

$$\begin{aligned} |P'| &\leq 2|P| + \sum_{i=1}^{\log_3 |P|} \ell_i(P) \frac{3^i}{\gamma(t)} \\ &= 2|P| + \sum_{i=1}^{\log_3 |P|} (1-\varepsilon) \cdot \ell_i(P) \frac{3^i}{\gamma(t)} + \varepsilon \cdot \ell_i(P) \frac{3^i}{\gamma(t)} \end{aligned}$$

We choose $\gamma(t) = \log_3(t)$. From Lemma 4.5 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 $t \geq 3^i \Rightarrow \log_3(t) \geq i$.

$$|P'| \leq 2|P| + \sum_{i=1}^{\log_3 |P|} (1-\varepsilon) \cdot \ell_i(P) \frac{3^i}{\frac{1}{2}\log_3(3^i \varepsilon \ell_i(P))} + \varepsilon \cdot \ell_i(P) \frac{3^i}{i}$$

As $x/\log(cx)$ grows monotonically for $x > e/c$,
from the bound for $\ell_i(P)$ from Lemma 4.4 follows:

$$\begin{aligned} &\leq 2|P| + \sum_{i=1}^{\log_3 |P|} \left((1-\varepsilon) \cdot \frac{\left(2\frac{|P|}{3^i} + 4\right) 3^i}{\frac{1}{2}\log_3\left(3^i \sigma\, \varepsilon \left(2\frac{|P|}{3^i} + 4\right)\right)} + \varepsilon \cdot \left(2\frac{|P|}{3^i} + 4\right) \cdot \frac{3^i}{i} \right) \\ &\leq 2|P| + \sum_{i=1}^{\log_3 |P|} \left((1-\varepsilon) \cdot \frac{2|P| + 4 \cdot 3^i}{\frac{1}{2}\log_3(2\sigma\, \varepsilon |P|)} + \varepsilon \cdot \frac{2|P| + 4 \cdot 3^i}{i} \right) \\ &\leq 2|P| + (1-\varepsilon) \cdot \left(\frac{4|P| \log_3 |P| + 8\sum_{i=1}^{\log_3 |P|} 3^i}{\log_3(2\sigma\, \varepsilon |P|)} \right) + \varepsilon \sum_{i=1}^{\log_3 |P|} \left(\frac{2|P| + 4 \cdot 3^i}{i} \right) \\ &\leq 2|P| + (1-\varepsilon) \cdot \left(\frac{4|P| \log_3 |P| + 12|P|}{\log_3(2\sigma\, \varepsilon |P|)} \right) + \varepsilon \cdot \left(2|P| \log_3 \log_3 |P| + 8|P|\right) \end{aligned}$$

Choose $\varepsilon = 1/\log_3 |P|; (1-\varepsilon) \leq 1$

$$\begin{aligned} &\leq 2|P| + \frac{4|P| \log_3 |P| + 12|P|}{\log_3(2\sigma\, |P| / \log_3 |P|)} + \frac{2|P| \log_3 \log_3 |P| + 8|P|}{\log_3 |P|} \end{aligned}$$

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

$$\begin{aligned} &\leq 2|P| + 16|P| \frac{\log_3 |P|}{\log_3(2\sigma) + \frac{1}{2}\log_3 |P|} + 10|P| \frac{\log_3 \log_3 |P|}{\log_3 |P|} \\ &\leq 44|P| \end{aligned}$$

■

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

4.6.5 Traffic Analysis

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 4.6.2). 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 4.10) 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 4.15). 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. Therefore, 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 $9g \times 9g$ square (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 4.6 *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.

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 4.16. If the small square "b" in case 1 in the figure is further subdivided, the subdivision rule requires a subdivision of square "A". The subdivision of this orthogonally neighboring square is already paid according to Payment Rule 1, since 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

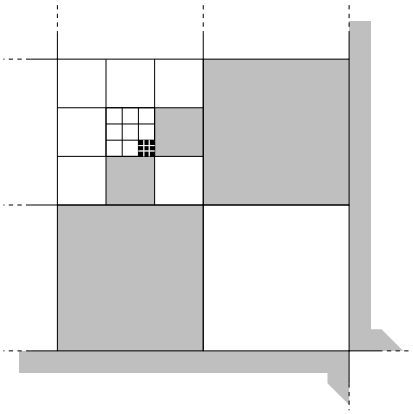


Figure 4.15: *Worst case for neighbor-induced subdivision: after the subdivision of the black square the shaded squares have to be subdivided, too.*

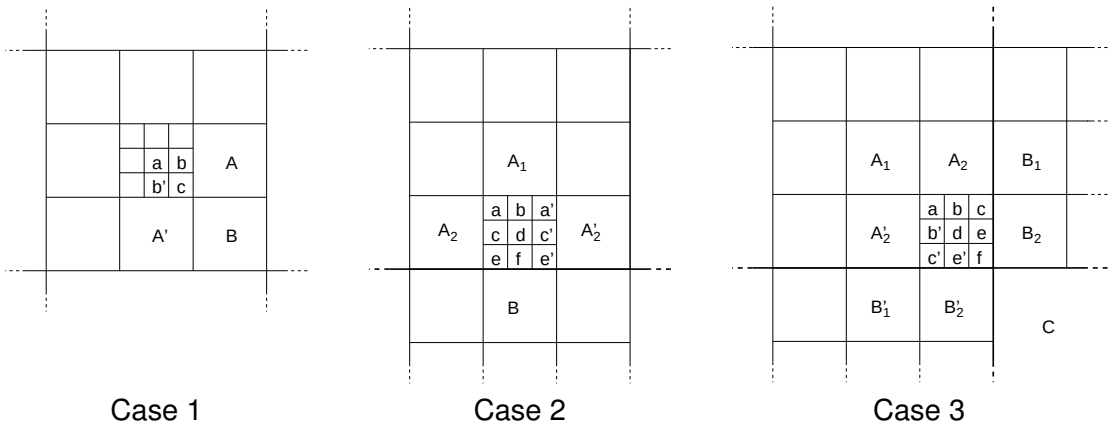


Figure 4.16: *Illustrations for the proof of Lemma 4.6. Symmetric cases are indicated by apostrophes.*

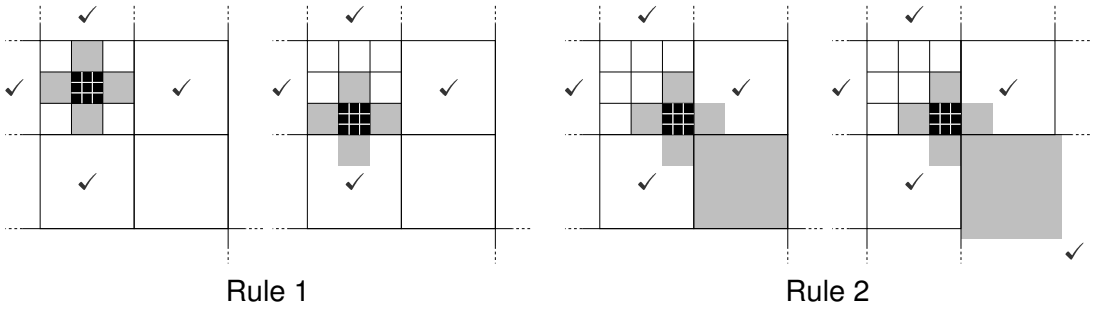


Figure 4.17: *Payment rules for barriers that cause a subdivision. A subdivision of the black square pays also the subdivision of the shaded squares. Marked squares (✓) are already paid.*

other cases are shown in the following table:

Subdivided square		Induced subdivision	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 the first subdivision rule. ■

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

Theorem 4.5 *The JITE 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 a square is denoted by $E(g) := \eta(1 + \frac{1}{\gamma})4g$, 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 4.6). The traffic for a square of size $g \times g$ with subdivision can be upper-bound as follows.

$$\begin{aligned}
\mathbf{Tr}(g \times g) &\leq E(g) + \sum_{i=1}^{\log_3 g} \frac{p \cdot \gamma(t)}{3^i}\, 126\, E(3^{i-1}) \\
&\leq \eta \left(1 + \frac{1}{\gamma(t)}\right) 4\, g + \sum_{i=1}^{\log_3 g} \frac{p \cdot \gamma(t)}{3^i}\, 126 \cdot \eta \left(1 + \frac{1}{\gamma(t)}\right) 4 \cdot 3^{i-1} \\
&\leq 8\, \eta \cdot g + p \log_3 g\, (\gamma(t) + 1)\, 168\, \eta \\
&\leq 168\, \eta\, \left(g + p \log_3 g\, (\log_3(t) + 1)\right)
\end{aligned}$$

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 4.9). 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}. Thus, in the i-th step there are 8 new squares of side length 3^{i-1} in each quadrant that 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.

$$
\begin{aligned}
\mathbf{Tr} &\leq 4\,\mathbf{Tr}(3\times 3) \;+\; 4\sum_{i=2}^{\lceil \log_3 d'\rceil} 8\,\mathbf{Tr}(3^i\times 3^i)\\
&\leq 4c\left(3+p_1\log_3 d\right) \;+\; 32\sum_{i=2}^{(\log_3 d')+1} c\left(3^i+p_i\log_3 3^i\log_3 d\right)\\
&\leq 32c\sum_{i=1}^{(\log_3 d')+1}\left(3^i+i\,p_i\,\log_3 d\right)\\
&\leq 144c\,d'+32c\log_3 d\sum_{i=0}^{(\log_3 d')+1} i\,p_i\\
&\leq 144c\,d'+32c\log_3 d\,(\log_3 d'+1)\,p\\
&= \mathcal{O}(d+p\log^2 d)
\end{aligned}
$$

■

Corollary 4.4 *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 $\mathcal{O}(\log^2 d)$.*

4.7 Conclusion and Outlook

Route discovery in faulty mesh networks is an online problem whose difficulty depends on the barriers in the network. Therefore we define comparative performance measures that compare the time and the traffic to the lower bound. The lower bound for the time is determined by the length of the shortest barrier-free path, whereas the online lower bound for traffic depends on the perimeters of the barriers. We have seen that basic strategies for online routing are *either* time-efficient *or* traffic-efficient but fail to optimize both measures. These strategies are outperformed by the JITE algorithm that solves the problem efficiently with respect to both time and traffic. It works asymptotically as fast as flooding and approaches the online lower bound for traffic up to a polylogarithmic factor. This result answers the question whether there are time-optimal strategies that decrease the traffic overhead of flooding algorithms significantly.

The following table summarizes the time and traffic performance as well as the combined comparative ratio $\mathcal{R}_c$ (Def. 4.5) of the presented algorithms.

Strategy	Time	Traffic	$\mathcal{R}_c$
Exp. Ring Search [JM96, PBD03]	$\mathcal{O}(d)$	$\mathcal{O}(d^2)$	$\mathcal{O}(d)$
Lucas' Algorithm [Luc88]	$\mathcal{O}(d+p)$	$\mathcal{O}(d+p)$	$\mathcal{O}(d)$
Alternating Algorithm	$\mathcal{O}(d^{3/2})$	$\mathcal{O}(\min\{d^2, d^{3/2}+p\})$	$\mathcal{O}(\sqrt{d})$
JITE Algorithm	$\mathcal{O}(d)$	$\mathcal{O}(d+p\log^2 d)$	$\mathcal{O}(\log^2 d)$
Online Lower Bound (cf. [BRS97])	$\Omega(d)$	$\Omega(d+p)$	$\Omega(1)$

The JITE algorithm combines several techniques. First, the search area is expanded continuously and thereby an adaptive grid of frames is constructed, which becomes denser in the proximity of barriers. On this grid a slowly proceeding shoreline (slow search) simulates a breadth-first search algorithm. This shoreline triggers the just-in-time exploration of new frames. 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 the perimeters of all barriers.

The efficiency of the JITE algorithm is due to the following ideas: The frames of the grid subdivision provide a constant-factor approximation of the shortest path. They are used by the slow search algorithm, which is slowed down by a constant factor. This way the optimal asymptotic time bound is reached. The traffic-efficiency is achieved by the grid subdivision which is only partially constructed in the proximity of the shoreline. The grid subdivision becomes denser in the vicinity of barriers. This is necessary to guarantee the constant-factor approximation of the shortest path, but it causes a logarithmic factor regarding the traffic. The second logarithmic factor is due to the detour that is allowed when traversing a frame and that prevents a large frame from being subdivided because of a small barrier.

Open problems

The JITE algorithm has a combined comparative ratio of $\mathcal{O}(\log^2 d)$ which is due to the traffic overhead. The question, whether this bound is tight or whether there is a small logarithmic trade-off between time and traffic remains still open. Furthermore, the time of the JITE algorithm is delayed by a large constant factor. It seems possible to decrease this factor without an asymptotic increase of the traffic.

The presented techniques could also be generalized to a routing algorithm for three-dimensional meshes. However, a simple generalization of the JITE algorithm would cause a significant increase in traffic. Thus, the problem of an efficient online routing in higher dimensions is wide open.

The model of a faulty mesh network is also similar to grid-based environment models, which are used in robot motion planning (see Section 2.5.2). However, it is not clear if some of the presented techniques can be used for distributed navigation algorithms, since parallelism, as it can be exploited in communication networks, has no equivalent in robotics scenarios where only a constant number of robots can be assumed.

5

Summary

In this thesis, the efficiency of position-based routing algorithms was studied. We began our considerations with the problem of position-based routing in wireless networks. These networks are usually modeled by unit disk graphs. Many known strategies rely on the geometric properties of these graphs, e.g. the face traversal algorithms. We abstracted from the graph topology issues and presented a position-based routing protocol, that is based on a grid subdivision of the plane. This protocol maps the communication links of the network and the void regions in between to link cells and barrier cells in the grid subdivision. This cell structure can be constructed locally without extra communication and provides an implicit planarization, which is required for loop-free routing in recovery mode. The advantage of this approach over graph planarization strategies is that no communication links are explicitly forbidden.

Besides these practical considerations, we have shown that routing on the cell structure is equivalent to routing in the original network. The cell structure corresponds to the model of a mesh network with faulty nodes. Based on this model, we have studied position-based routing algorithms under the aspect of routing time and traffic, i.e. the number of produced messages. For the analysis we use comparative measures that compare the time and the traffic of the algorithm with the lower bounds for online algorithms. The known approaches are either time-efficient or traffic-efficient, but fail to optimize both. As single-path routing strategies are not able to reach the lower time bound, parallelism is inevitable for time-optimal online routing. But is it necessary to flood the network in order to optimize the routing time? The answer to this question is given in this thesis. We presented the JITE algorithm that delivers a message in time $\mathcal{O}(d)$ with traffic $\mathcal{O}(d + p \log^2 d)$ where d is the length of the shortest path and p the number of border nodes being adjacent to faulty nodes. This algorithm works asymptotically as fast as flooding, which is the best possible strategy when optimizing routing time. But in contrast to flooding, this algorithm is nearly traffic-optimal up to a poly-logarithmic factor.

Further work on this routing problem includes the question for efficient routing in three-dimensional meshes and the question whether the obtained bounds are tight or whether there is a small logarithmic trade-off between time and traffic.

Bibliography

[Ad82] Harold Abelson and Andrea diSessa. *Turtle Geometry*. MIT Press, 3rd edition, 1982.

[AH00] Susanne Albers and Monika Rauch Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000.

[AKS02] Susanne Albers, Klaus Kursawe, and Sven Schuierer. Exploring unknown environments with obstacles. *Algorithmica*, 32(1):123–143, January 2002.

[ALW+03] Khaled Alzoubi, Xiang-Yang Li, Yu Wang, Peng-Jun Wan, and Ophir Frieder. Geometric spanners for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):408–421, 2003.

[AWZ00] Dana Angluin, Jeffery Westbrook, and Wenhong Zhu. Robot navigation with distance queries. *SIAM Journal on Computing*, 30(1):110–144, 2000.

[BBF+96] Piotr Berman, Avrim Blum, Amos Fiat, Howard Karloff, Adi Rosén, and Michael Saks. Randomized robot navigation algorithms. In *Proceedings of the 7th annual ACM-SIAM Symposium on Discrete algorithms (SODA '96)*, pages 75–84, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.

[BBFY94] Eldad Bar-Eli, Piotr Berman, Amos Fiat, and Peiyuan Yan. Online navigation in a room. *Journal of Algorithms*, 17(3):319–341, 1994.

[BCR93] Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.

[BCSW98] Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward. A distance routing effect algorithm for mobility (DREAM). In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 76–84. ACM Press, 1998.

[BDEK06] Prosenjit Bose, Luc Devroye, William Evans, and David Kirkpatrick. On the spanning ratio of gabriel graphs and beta-skeletons. *SIAM Journal on Discrete Mathematics*, 20(2):412–427, 2006.

[BE98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[Beb06] Inés Bebea González. Geographic routing using a cell structure in wireless ad-hoc networks. Master's thesis, Universidad Carlos III de Madrid, Spain, 2006. to appear.

[Ber98] Piotr Berman. On-line searching and navigation. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 232–241. Springer, 1998.

[BFNO03] Lali Barrière, Pierre Fraigniaud, Lata Narayanan, and Jaroslav Opatrny. Robust position-based routing in wireless ad hoc networks with irregular transmission ranges. *Wireless Communications and Mobile Computing*, 3(1):141–153, March 2003.

[BGI92] Reuven Bar-Yehuda, Oded Goldreich, and Alon Itai. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *Journal of Computer and System Sciences*, 45(1):104–126, August 1992.

[BHSS03] Brian Blum, Tian He, Sang Son, and John Stankovic. IGF: A state-free robust communication protocol for wireless sensor networks. Technical Report CS-2003-11, University of Virginia, USA, April 2003.

[BK78] Manuel Blum and Dexter Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS'78)*, pages 132–142, 1978.

[BM04a] Prosenjit Bose and Pat Morin. Competitive online routing in geometric graphs. *Theoretical Computer Science*, 324(2-3):273–288, September 2004.

[BM04b] Prosenjit Bose and Pat Morin. Online routing in triangulations. *SIAM Journal on Computing*, 33(4):937–951, May 2004.

[BMSU01] 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.

[BR06] Inés Bebea González and Stefan Rührup. The cell-based geographic forwarding protocol, Specification. Technical Report TR-RSFB-06-79, University of Paderborn, 2006.

[Bra03] Ulrike Brandt. Wie fängt man Mäuse? Technical Report AFS-3/03, Technische Universität Darmstadt, October 2003.

[BRS97] Avrim Blum, Prabhakar Raghavan, and Baruch Schieber. Navigating in unfamiliar geometric terrain. *SIAM Journal on Computing*, 26(1):110–137, February 1997.

[Bud75] Lothar Budach. On the solution of the labyrinth problem for finite automata. *Elektronische Informationsverarbeitung und Kybernetik*, 11(10-12):661–672, 1975.

[CCT03] Chih-Yung Chang, Chao-Tsun Chang, and Shin-Chih Tu. Obstacle-free geocasting protocols for single/multi-destination short message services in ad hoc networks. *Wireless Networks*, 9(2):143–155, 2003.

[CL03] Tracy Camp and Yu Liu. An adaptive mesh-based protocol for geocast routing. *Journal of Parallel and Distributed Computing*, 63(2):196–213, 2003.

[CMS97] 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.

[Coy77] Wolfgang Coy. Automata in labyrinths. In *Fundamentals of Computation Theory, Proceedings of the 1977 International FCT-Conference (FCT), LNCS 56*, pages 65–71. Springer-Verlag, September 1977.

[CSU95] Danny Z. Chen, Robert J. Szczerba, and John J. Uhran, Jr. Determining conditional shortest paths in an unknown 3-d environment using framed-octrees. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 4101–4106, October 1995.

[CSU97] Danny Z. Chen, Robert J. Szczerba, and John J. Uhran, Jr. A framed-quadtree approach for determining euclidean shortest paths in a 2-d environment. *IEEE Transactions on Robotics and Automation*, 13(5):668–681, October 1997.

[DKP98] Xiaotie Deng, Tiko Kameda, and Christos Papadimitriou. How to learn an unknown environment. I: The rectilinear case. *Journal of the ACM*, 45(2):215–245, 1998.

[DP99] Xiaotie Deng and Christos H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.

[FGG04] Qing Fang, Jie Gao, and Leonidas J. Guibas. Locating and bypassing routing holes in sensor networks. In *Proceedings of the 23rd Conference of the IEEE Communications Society (INFOCOM'04)*, March 2004.

[Fin87] Gregory G. Finn. Routing and addressing problems in large metropolitan-scale internetworks. Technical Report ISI/RR-87-180, University of Southern California, March 1987.

[FWMH03] Holger Füßler, Jörg Widmer, Martin Mauve, and Hannes Hartenstein. A novel forwarding paradigm for position-based routing (with implicit addressing). In *Proceedings of the IEEE 18th Annual Workshop on Computer Communications (CCW 2003)*, pages 194–200, October 2003.

[Gal80] Shmuel Gal. *Search Games*, volume 149 of *Mathematics in Science and Engineering*. Academic Press, 1980.

[GGH+01] Jie Gao, Leonidas J. Guibas, John E. Hershberger, Li Zhang, and An Zhu. Geometric spanner for routing in mobile networks. In *Proc. of the 2nd Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'01)*, pages 45–55. ACM, Oct 2001.

[GS69] K. Ruben Gabriel and Robert R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18(3):259–278, September 1969.

[GS03] Silvia Giordano and Ivan Stojmenovic. Position-based ad hoc routes in ad hoc networks. In Mohammad Ilyas, editor, *The Handbook of Ad Hoc Wireless Networks*, chapter 16. CRC Press, 2003.

[GS04] Silvia Giordano and Ivan Stojmenovic. Position based routing algorithms for ad hoc networks: A taxonomy. In Xiuzhen Cheng; Xiao Huang; Ding-Zhu Du, editor, *Ad Hoc Wireless Networking*, pages 103–136. Kluwer, 2004.

[HA92] Yong K. Hwang and Narendra Ahuja. Gross motion planning – a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[HB03] Marc Heissenbüttel and Torsten Braun. A novel position-based and beacon-less routing algorithm for mobile ad-hoc networks. In *Proc. of the 3rd IEEE Workshop on Applications and Services in Wireless Networks (ASWN' 03)*, pages 197–209, Bern, Switzerland, July 2003.

[HBBW04] Marc Heissenbüttel, Torsten Braun, Thomas Bernoulli, and Markus Wälchli. BLR: Beacon-less routing algorithm for mobile ad-hoc networks. *Computer Communications*, 27(11):1076–1086, July 2004.

[Hem77] Armin Hemmerling. *Labyrinth Problems – Labyrinth-Searching Abilities of Automata*, volume 144 of *Teubner-Texte zur Mathematik*. Teubner, Leipzig, 1977.

[Hem93] Armin Hemmerling. Navigation without perception of coordinates and distances. Technical Report TR-93-018, International Computer Science Institute, Berkeley, CA, 1993.

[HL86] Tingh-Chao Hou and Victor Li. Transmission range control in multihop packet radio networks. *IEEE Transactions on Communications*, 34(1):38–44, January 1986.

[HS99] John Hershberger and Subhash Suri. An optimal algorithm for euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

[IKKL05] Christian Icking, Tom Kamphans, Rolf Klein, and Elmar Langetepe. Exploring simple grid polygons. In *Proceedings of the 11th Annual International Conference on Computing and Combinatorics, (COCOON 2005), LNCS 3595*, pages 524–533. Springer, 2005.

[JC02] Xia Jiang and Tracey Camp. Review of geocasting protocols for a mobile ad hoc network. In *Proceedings of the Grace Hopper Celebration (GHC)*, 2002.

[JM96] 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.

[JS01] Patrick Jaillet and Matthew Stafford. Online searching. *Operations Research*, 49(4):501–515, 2001.

[JT92] Jerzy W. Jaromczyk and Godfried T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80:1502–1517, 1992.

[Kar00] Brad Karp. *Geographic Routing for Wireless Networks*. PhD thesis, Harvard University, Cambridge, MA, October 2000.

[KGKS05a] Young-Jin Kim, Ramesh Govindan, Brad Karp, and Scott Shenker. Geographic routing made practical. In *Proceedings of the 2nd USENIX/ACM Symposium on Networked System Design and Implementation (NSDI'05)*, pages 217–230, May 2005.

[KGKS05b] Young-Jin Kim, Ramesh Govindan, Brad Karp, and Scott Shenker. On the pitfalls of geographic routing. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications – Principles of Mobile Computing (DIALM-POMC'05)*, pages 34–43, September 2005.

[KK00] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)*, pages 243–254, 2000.

[Kle94] Jon M. Kleinberg. On-line search in a simple polygon. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '94)*, pages 8–15, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[KP00] Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300–317, 2000.

[KRR98] Ishay Kamon, Elon Rimon, and Ehud Rivlin. Tangentbug: A range-sensor-based navigation algorithm. *The International Journal of Robotics Research*, 17(9):934–953, September 1998.

[KRT93] Ming-Yang Kao, John H. Reif, and Stephen R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, 1993.

[KSU99] Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia. Compass routing on geometric networks. In *Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG'99)*, pages 51–54, Vancouver, August 1999.

[KV00] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. *Wireless Networks*, 6(4):307–321, 2000.

[KW05] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. Wiley, 2005.

[KWZ02] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Asymptotically optimal geometric mobile ad-hoc routing. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM)*, pages 24–33. ACM Press, 2002.

[KWZZ03] Fabian Kuhn, Roger Wattenhofer, Yan Zhang, and Aaron Zollinger. Geometric ad-hoc routing: Of theory and practice. In *Proceedings of the 22th ACM Symposium on Principles of Distributed Computing (PODC'03)*, pages 63–72, 2003.

[Lat93] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 3rd edition, 1993.

[LaV06] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[LJC+00] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)*, pages 120–130, New York, NY, USA, 2000. ACM Press.

[LLM06] Ben Leong, Barbara Liskov, and Robert Morris. Geographic routing without planarization. In *Proceedings of the 3rd USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '06)*, 2006.

[LLS00] W.-H. Liao, Y.-C-Tsengand K.-L. Lo, and J.-P. Sheu. GeoGRID: A geocasting protocol for mobile ad hoc networks based on GRID. *Journal of Internet Technology*, 1(2):23–32, 2000.

[LS84] Vladimir J. Lumelsky and Alexander A. Stepanov. Effect of uncertainty on continuous path planning for an autonomous vehicle. In *Proceedings of the 23rd IEEE Conference on Decision and Control*, 1984.

[LS86] Vladimir J. Lumelsky and Alexander A. Stepanov. Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE Transactions on Automatic Control*, 31(11):1058–1063, November 1986.

[LS87] 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.

[LS90] Vladimir J. Lumelsky and Tim Skewis. Incorporating range sensing in the robot navigation function. *IEEE Transactions on Systems, Man and Cybernetics*, 20(5):1058 – 1069, 1990.

[LTS01] W.-H. Liao, Y.-C. Tseng, and J.-P. Sheu. Grid: A fully location-aware routing protocol for mobile ad hoc networks. *Telecommunication Systems*, 18(1):37–60, 2001.

[Luc88] Caro Lucas. Comments on "dynamic path planning for a mobile automaton with limited information on the environment". *IEEE Transactions on Automatic Control*, 33(5):511, May 1988.

[Luc92] Édouard Lucas. Le jeu des labyrinthes. In *Récréations Mathématiques*, volume 1, chapter 3, pages 41–55. Blanchard, Paris, (1882) reprint 1992.

[Lum87] Vladimir J. Lumelsky. Algorithmic and complexity issues of robot motion in an uncertain environment. *Journal of Complexity*, 3(2):146–182, 1987.

[Lum91] Vladimir J. Lumelsky. A comparative study on the path length performance of maze-searching and robot motion planning algorithms. *IEEE Transactions on Robotics and Automation*, 7(1):57–66, February 1991.

[Mai04] Christian Maihöfer. A survey of geocast routing protocols. *IEEE Communications Surveys and Tutorials*, 6(2):32–42, 2004.

[MMS88] Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive algorithms for on-line problems. In *Proceedings of the 20th annual ACM symposium on Theory of computing (STOC'88)*, pages 322–333, New York, NY, USA, 1988. ACM Press.

[MWH01] Martin Mauve, Jörg Widmer, and Hannes Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network Magazine*, 15(6):30–39, November 2001.

[NHN03] Ryo Nogami, Satoru Hirao, and Hiroshi Noborio. On the average path lengths of typical sensor-based path-planning algorithms by uncertain random mazes. In *Proceeding of the IEEE International Symposium on Computational Intelligence on Robotics and Automation (CIRA'03)*, pages 471–478, 2003.

[NNH04] Hiroshi Noborio, Ryo Nogami, and Satoru Hirao. A new sensor-based path-planning algorithm whose path length is shorter on the average. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'04)*, pages 2832–2839, 2004.

[NY98] Hiroshi Noborio and Takashi Yoshioka. Sensor-based navigation of a mobile robot under uncertain conditions. In Kamal Gupta and Angel P. del Pobil, editors, *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pages 325–347. John Wiley & Sons, 1998.

[PBD03] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir Das. Ad hoc on-demand distance vector (AODV) routing. IETF RFC 3561, July 2003.

[PY89] Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. In *Proceedings of the 16th International Colloquium on Automata, Languages, and Programming (ICALP'89)*, pages 610–620. Elsevier Science Publishers Ltd., 1989.

[PY91] Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

[RKSI93] N. Rao, S. Kareti, W. Shi, and S. Iyengar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical Report ORNL/TM-12410, Oak Ridge National Laboratory, jul 1993. 1993.

[RKY+02] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. Ght: A geographic hash table for data-centric storage in sensornets. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, September 2002.

[RPSS03] Ananth Rao, Christos Papadimitriou, Scott Shenker, and Ion Stoica. Geographic routing without location information. In *Proceedings of the 9th annual international conference on Mobile computing and networking (MobiCom '03)*, pages 96–108, New York, NY, USA, 2003. ACM Press.

[RS05a] Stefan Rührup and Christian Schindelhauer. Competitive time and traffic analysis of position-based routing using a cell structure. In *Proceedings of the 5th IEEE International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (IPDPS/WMAN'05)*, page 248, 2005.

[RS05b] Stefan Rührup and Christian Schindelhauer. Online routing in faulty meshes with sub-linear comparative time and traffic ratio. In *Proceedings of the 13th European Symposium on Algorithms (ESA'05), LNCS 3669*, pages 23–34. Springer-Verlag, 2005.

[RS06] Stefan Rührup and Christian Schindelhauer. Online multi-path routing in a maze. In *Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC'06)*, 2006. to appear.

[Shi00] Zvi Shiller. On-line sub-optimal obstacle avoidance. *International Journal of Robotics Research*, 19(5):480–497, May 2000.

[Ste94] Anthony Stentz. Optimal and efficient path planning for partially-knownenvironments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'94)*, pages 3310–3317, 1994.

[Ste01] Angelika Steger. *Diskrete Strukturen (Band 1)*. Springer-Verlag, 2001.

[Sto04] Ivan Stojmenovic. Geocasting with guaranteed delivery in sensor networks. *Wireless Communications*, 11(6):29–37, 2004.

[Sto06] Ivan Stojmenovic. Geocasting in ad hoc and sensor networks. In Jie Wu, editor, *Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks*, pages 79–97. Auerbach Publications, 2006.

[SV90a] A. Sankaranarayanan and M. Vidyasagar. A new path planning for moving a point object admidst unknown obstacles in a plane. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA'90)*, pages 1930–1936, 1990.

[SV90b] A. Sankaranarayanan and M. Vidyasagar. Path planning for moving a point object admidst unknown obstacles in a plane: A new algorithm and a general theory for algorithm development. In *Proceedings of the 29th IEEE Conference on Decision and Control*, pages 1111–1119, 1990.

[TK84] Hideaki Takagi and Leonard Kleinrock. Optimal transmission ranges for randomly distributed packet radio terminals. *IEEE Transactions on Communications*, 32(3):246–257, March 1984.

[Tou80] Godfried T. Toussaint. The relative neighborhood graph of a finite planar set. *Pattern recognition*, 12(4):261–268, 1980.

[WJ02] Jie Wu and Zhen Jiang. Extended minimal routing in 2-d meshes with faulty blocks. In *Proceedings of the 1st International Workshop on Assurance in Distributed Systems and Applications (in conjuction with ICDCS 2002)*, pages 49–55, 2002.

[WL02] Yu Wang and Xiang-Yang Li. Distributed spanner with bounded degree for wireless ad hoc networks. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS'02)*, page 120. IEEE Computer Society, 2002.

[Wu00] 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.

[YSSB98] Alex Yahja, Anthony Stentz, Sanjiv Singh, and Barry Brummit. Framed-quadtree path planning for mobile robots operating in sparse environments. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA'98)*, Leuven, Belgium, May 1998.

[ZK98] Lev Zakrevski and Mark Karpovsky. Fault-tolerant message routing for multiprocessors. In *Parallel and Distributed Processing, LNCS 1388*, pages 714–730. Springer, 1998.

[ZR03a] Michele Zorzi and Ramesh R. Rao. Geographic random forwarding (GeRaF) for ad hoc and sensor networks: Energy and latency performance. *IEEE Transactions on Mobile Computing*, 02(4):349–365, 2003.

[ZR03b] Michele Zorzi and Ramesh R. Rao. Geographic random forwarding (GeRaF) for ad hoc and sensor networks: Multihop performance. *IEEE Transactions on Mobile Computing*, 02(4):337–348, 2003.

Index