Arithmetic Geographical Coding And Routing

M.Sc. Thesis

Submitted by: Mr. Yaniv Dvory

Advisors: Dr. Chen Avin, Prof. Ran Giladi

Department of Communication Systems Engineering

Faculty of Engineering

Ben-Gurion University of the Negev

October 1, 2010
Acknowledgements

I would like to show my deepest gratitude to my supervisor, Dr. Chen Avin, whose passion for research, guidance and support from the initial to the final level enabled me to develop an understanding of the subject and made this thesis possible.

I would also like to express my gratitude to my professors for their contribution to this work in giving me different ideas and inspiring me in their classes and in the conversations we had. Especially Dr. Zvi Lotker, Dr. Omer Gurewitz and Prof. Ran Giladi.

Lastly, I would like to thank my family and girlfriend for their encouragement, and their contribution to my decision to take this adventure. This thesis would not have been possible without the support and encouragements of my beautiful girlfriend Erica Newman, who helped me at each and every step along the way.

Yaniv Dvory
Abstract

In this thesis we study two research aspects of geographical routing: efficient geographical addressing and a new routing scheme.

In the first part we offer a novel geographical routing algorithm that relies on a well known data structure called Quadtree. Quadtree is an efficient method of mapping a two-dimensional area by recursively partitioning it to disjoint squares. We present a greedy, guaranteed delivery routing algorithm called Greedy-Quadtree-Greedy (GQG). The algorithm is robust to dynamics in the non-Quadtree edges and overcomes local minimums without the use of planarization, face routing, or searching. GQG is a tree-based routing algorithm; it makes greedy forwarding based on the location information that is extracted from the Quadtree addresses of the nodes. Bypassing voids is done by a concept of "tree routing with shortcuts", which can significantly improve hop stretch and load balancing. As part of the routing system, we present three algorithms: address distribution, network topology discovery, and geographical routing with guaranteed delivery. We keep all broadcasts bounded to one hop, and the nodes’ routing state depends on their degree rather than the overall network size. We prove the correctness of the algorithms and present simulations that show the protocol improvement over simple tree-based routing in load balancing and routing stretch.

In the second part we present a bit efficient arithmetic geographical coding system that extends the coding system of Quadtree. Each codeword is a node’s geographical address and
therefore has to be unique, short, and contain information about the node’s location in the
unit square. Our Arithmetic Geographical Address system (AGC) idea is to represent each
node by a axis parallel rectangle in the unit square, such that each rectangle contains one
node only. The code word given to a node is a two-field binary address indicating the x and
y coordinates of the bottom left corner of the rectangle, where the length of each rectangle
side is indicated in the length of the corresponding address field. We provide an address
distribution algorithm for any set of nodes located in a unit square.

We prove the correctness of our algorithm and present simulations that show that the
address system described improves Quadtrees bit efficiency.
Contents

1 Introduction 10

2 Related Work 16

3 Preliminaries 23

1 Geographical Quadtree Routing 30

4 Problem Definition and Routing Idea 31
   4.1 Quadtree Geographical Address 32
   4.2 MultiQuadtree Network 34
   4.3 Quadtree-Based Routing Algorithms 37

5 GQG Routing Algorithm 40
   5.1 Quadtree Routing with Shortcuts 40
   5.2 GQG: Greedy-Quadtree-Greedy Routing Algorithm 42
   5.3 Algorithm Correctness 44
   5.4 Routing Example 45

6 MQD MultiQuadtree Network Discovery 47
   6.1 Algorithm Idea 48
13 Conclusions and Future Work
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Quadtree partitioning (right) and the corresponding Quadtree graph (left)</td>
<td>13</td>
</tr>
<tr>
<td>3.1</td>
<td>Tree graph example</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Face routing using &quot;right hand rule&quot;</td>
<td>28</td>
</tr>
<tr>
<td>4.1</td>
<td>Quadtree of 17,168 weather stations around the world from Mathematica 7</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>Quadtree address is built of suffixes added with each recursive partition of</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>the square area according to the suffix table (top left). A Quadtree</td>
<td></td>
</tr>
<tr>
<td></td>
<td>partition example for a set of nodes (right) and some address examples (bot-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tom left)</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>MultiQuadtree network example: (A) $G(V, E)$ and $Quadtree(V)$. (B) The</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>MultiQuadtree hierarchy. (C) $Qadd$ and $TSet$ table</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Routing example - MultiQuadtree infrastructure is marked in solid lines and</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>additional edges in dashed lines</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Example of population maps of nodes from the network shown in Fig. 4.3: (A)</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>The initial population map of node $d$, (B) The initial population map of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>node $b$, and (C) The population map of node $d$ after receiving and learning</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the population map of node $b$ after executing $LearnMap_d(b)$</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>MultiQuadtree network, MQD algorithm correctness proof</td>
<td>53</td>
</tr>
<tr>
<td>8.1</td>
<td>400 nodes, $D=10$, Stretch and found path length for the different graphs</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>and routing algorithms</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>400 nodes, $D=10$, $T=1$ Accumulating Load balance</td>
<td>63</td>
</tr>
<tr>
<td>8.3</td>
<td>400 nodes, $D=10$, Distance Related graph, Pure Tree/GQG routing load</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>balance ratio</td>
<td></td>
</tr>
<tr>
<td>8.4</td>
<td>$D=10$, Average found path length for different algorithms as a function of</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>the network size</td>
<td></td>
</tr>
</tbody>
</table>
10.1 Arithmetic Geographical Address example

10.2 Symmetric arithmetic address

11.1 Two-bit matching to \( v \) partitioning of the unit square. Nodes in the white areas will have values matching \( v \) according to this partition in the corresponding field. Nodes in the dashed areas will have matching values of 3 or more in at least one of the fields.

11.2 The unit square partitioned by matching values to node \( v \) (left) and the Dplane of node \( v \) with the Dpoints corresponding to the neighboring nodes (right).

11.3 Contradicting situation of legal rectangle \( R \) in \( \text{Dplane}(v) \) (left) and illegal \( C(R) \) in the unit square (right).

11.4 AGC algorithm correctness proof

12.1 32 uniformly located nodes coded in Quadtree and AGC coding systems. The areas in the unit square are represented by the nodes code words and a histogram of the code.

12.2 512 normally located nodes coded in Quadtree and AGC coding systems, the areas in the unit square represented by the nodes code words, and a histogram of the code.

12.3 17,168 world wide weather stations nodes coded in Quadtree and AGC coding systems, the areas in the unit square represented by the nodes code words, and a histogram of the code.

12.4 Average code word length for a growing set of nodes

12.5 Maximal code word length for a growing set of nodes

12.6 Improvement in maximal code word length when using AGC
Chapter 1

Introduction

Communication networks, including data center and the Internet, continue to grow in size as the devices get smaller and less costly. These devices are taking a more dominate role in our daily lives and we are therefore faced with networks that contain much larger quantities of routing elements.

A basic demand from any routing algorithm is to guarantee the delivery of a message from point A to point B in the network. In today’s standard routing algorithms, the routing elements hold large routing tables that grow with the increasing size of the network. Constructing and maintaining these tables demands large memory space, lookup time, high computing capabilities, and a message exchanging process that creates large overhead traffic.

The idea of geographical routing algorithms is to take advantage of geographical information the routing elements have about the geographical location of their neighbors and themselves, in addition to information about the physical location of the routed message destination.

In a greedy geographical algorithm, each node forwards the message to one of its neighbors that is geographically closest to the destination. This way the nodes can deliver the message efficiently toward the destination, while maintaining small state and having only
local information about the network. This method greatly reduces memory use and lookup
time, and therefore scales much better. Such algorithms depend on geographical location
systems, which have recently become less costly, more available, and accurate with the de-
velopment and spread of GPS and other positioning systems such as Cricket [1].

The greedy geographical routing method is very simple and scaleable; however it does
not guarantee delivery of the messages. A local minimum, also known as void, is a node that
can no longer pass the message toward the destination in a greedy manner. This happens
when the node is closest to the destination among its neighbors, but is not the destination
node. This is the main difficulty with geographical routing algorithms. There are various
approaches to overcome this problem.

The first system that was developed to bypass local minimums relied on wide or
bounded message flooding, [2, 3, 4, 5, 6], which caused a significant amount of overhead,
and in some cases did not guarantee delivery.

The first geographic routing algorithm that achieved guaranteed delivery was initially
called compass routing [7], and was later called Face routing. Face routing is a very popular
method to bypass voids and is used in many of the existing geographic routing algorithms
[8, 9, 10, 11, 12] in different variations, which gradually improved the routing hop efficiency.
The hop efficiency of a routing algorithm, also known as hop stretch, is the ratio measured
between the number of nodes the message passes on the path to its destination found by
the algorithm, compared to the number of nodes the message passes in the shortest possible
route.

Face routing works only in planar graphs and might be very inefficient in route selec-
tions. Planar graphs are graphs that can be drawn on a two-dimensional surface with no
edge crossings. Planarization in a distributed manner works nicely in some graph models
such as unit disk graphs, but in general settings, planarization is not always possible, and
when it is, it could still be complex and costly, such as in CLDP [13].

Some of the algorithms start routing the messages in greedy mode, use face routing
in order to bypass voids, and switch back to greedy mode after passing the void, i.e., when
the packet reaches a node closer to the destination than the last local minimum. These
algorithms are called G-F-G, Greedy Face Greedy routing algorithms, e.g., [9].

Another system to bypass voids is presented in the paper Geographic Routing without
Planarization [14]. Leong et al. offered GDSTR, a tree-based guaranteed delivery geographical
routing algorithm. The tree is built such that each node is the root of an area containing
its sub-tree. The reason GDSTR is able to guarantee the delivery of packets in a connected
network is that the tree traversal forwarding mode alone is guaranteed to deliver the packet
to any node in the network when greedy forwarding fails. One complication of the proposed
method is in cases where the destination lies in the intersection of the areas rooted by two
or more nodes. In this case, packet delivery is guaranteed by systematic depth first search
in all the subtrees with areas containing the destination.

In geographical routing algorithms the nodes receive geographical addresses rather
than network addresses, like IP addresses. The geographical addresses of the nodes contain
information about their physical location. These addresses are stored in the routing tables
of the nodes and the source/destination fields of the messages, therefore, are preferred to be
as short as possible. The addresses of the nodes contain all known available geographical
information; therefore there is great importance in constructing them in such a way that
they contain a sufficient quantity of information for the routing algorithm.

Other existing geographical routing algorithms use preprocessing of the network topology with different methods [15] [16] [17] [18] [19]. The preprocessing stage is done in order
to calculate and distribute virtual coordinates for the nodes in the network. The virtual
coordinates are built in order to increase convexity and significantly reduce the probability
of the packet reaching a local minimum.
Overview and Main Contributions

In the first part we adopt a similar approach to GDSTR [14] and develop a distributed,
guaranteed delivery, geographic routing algorithm that uses tree routing where greedy routing
fails. The data structure we use to construct the geographical addresses of the nodes, as well
as for defining the hierarchy of the nodes used in the routing algorithm is called Quadtree
[20]. Quadtree is a well known data structure that has many applications in data clustering,
computational geometry, image processing, and more. The basic idea of Quadtree is to cover
a plane region of interest by a square, and then recursively partition it into four smaller
squares until each square contains a single point of interest, which, in our case, is simply a
node. The Quadtree addresses represent the squared areas contained in the unit square, in a
way that will be explained at a later point. We are not aware of previous usage of Quadtree
in the context of geographical routing.

![Figure 1.1: Quadtree partitioning (right) and the corresponding Quadtree graph (left)](image)

Based on a Quadtree, which provides hierarchical partitioning of the network into areas
with no overlap, we define geographical addresses and a tree. This tree allows us to route
packets with no routing errors or local searches. Moreover, our addressing enables the use of
unique and relatively short geographical addresses that reduce the overhead of each message
and the size of the routing tables. The length of the addresses depends on the density of the
nodes.

The routing system we develop, Greedy-Quadtree-Greedy (GQG), takes advantage of the stateless hop efficient greedy routing, as well as the guaranteed delivery of tree-based routing. A message is routed throughout the network in a greedy manner until it gets to a local minimum then switches to tree mode routing. The packet will return to greedy mode after passing the local minimum. To increase route efficiency and to achieve better load balancing, we extend the Quadtree into a MultiQuadtree network infrastructure that can support several roots. In addition, we allow shortcuts during the tree mode routing (route on edges that are not part of the tree). These shortcuts provide a sort of geographical routing on the tree but they avoid local minimum, reduce the stretch of the route, and increase load balancing.

We provide algorithms for MultiQuadtree network discovery (MQD), Quadtree address distribution (QAD), and geographical routing with guaranteed delivery (GQG). We prove the correctness of our algorithms and present simulations that show this routing system has very low stretch (i.e., the routes that the algorithm finds are close to the shortest routes in the network) and good load balancing.

In the second part we define a bit efficient arithmetic geographical address system for representing a set of nodes in the unit square. The basic structure we use to construct geographical addresses to nodes is a Quadtree.

As previously stated, geographical address must contain sufficient geographical information; however, it is preferred that they are short as they are contained in each source/destination message field and in the routing tables of the nodes. The message fields and the memory allocated for the routing table is determined according to the longest address because they have to be sufficient for all addresses. Therefore, it is important to minimize the maximal address length, while still maintaining sufficient information in the addresses set. This problem, to the best of our knowledge, has not been previously studied. Most of existing research on
geographical routing algorithms assumes the nodes addresses to be their GPS coordinates.

We provide an extension to the Quadtree system by allowing two main relaxations. The first is allowing rectangular areas rather than only squared ones. The address length of a node in Quadtree depends on the side length of its square, where smaller squares demand more bits for representation. Therefore, by allowing rectangular areas we allow the enlargement of one of the sides, which saves bits in the address.

The second relaxation is allowing the rectangular areas to overlap as long as there is still only one node in each rectangle; in other words, two or more rectangles can include the same area as long as it is clear of nodes, which leads to enlargement of the rectangles perimeter, thereby saving bits in the address.

We provide a detailed algorithm for finding the most efficient code for a set of nodes in the unit square. We prove the correctness of the algorithm and present simulations that show this code system greatly improves the Quadtree code system maximal code word length.
Chapter 2

Related Work

Previous research about geographical routing has concentrated on a few families of algorithms. We review the most influential groups of algorithms.

The first family of geographical routing algorithms \cite{2,3,4,5,6} uses some kind of limited flooding in order to bypass local minimums. This allows routing the message towards the target with less complexity. However, it increases the overhead caused by the algorithms, and does not guarantee delivery.

Young-Bae Ko and Nitin H. Vaidya present two algorithms for geocasting \cite{5}. Geocasting is sending a message to a group that consists of all nodes within a specific geographical area. Their algorithm uses flooding of the message to all nodes within a limited area. This is in contrast to using standard flooding, in which the message would be delivered to all reachable nodes, but causes greater overhead. The proposed approach is called "location-based multicast", as it makes use of location-based multicast groups and utilizes location information to reduce multicast delivery overhead.

Consider a node S that needs to multicast a message to all nodes that are currently located within a certain geographical region - the "Multicast Region". The multicast region
would be represented by some closed polygon such as a circle or a rectangle. Node S defines (implicitly or explicitly) a "Forwarding Zone" for the multicast message. Any node that receives the message forwards it only in the case that it belongs to the forwarding zone. The forwarding zone must include the multicast region and a path from S to multicast group members. Thus, there exists a trade-off between accuracy and the overhead of multicast delivery.

In Stefano Basagni, Irnrich Chlamtac, Violet R. Syrotiuk, and Barry Woodwards article [6], they base the routing algorithm on the angle of the receiver from the sender’s node. Forwarding the message is done by using flooding to all neighbors located in an $\alpha$ angle range to each side from the line connecting the sender and destination.

The algorithm defines sending management messages from each node telling its location, where messages from farther nodes will be received less frequently than messages from closer ones - the "Distance Effect". This is done by giving low TTL in most of the messages and high TTL for some of them. Frequency of sending these messages will be according to the speed of the node, fast moving nodes will send the messages more frequently than slow ones - the "Speed Effect".

Each node holds a table of all other nodes and their location according to the messages received from them and the time stamp of receiving the message.

The angle $\alpha$ is determined according to the speed and location of the target node, from which we can get the radius of the circle the destination should be located in. In case the sender is in the circle, $\alpha = \pi$ and the message will be sent to all neighbors. When the greedy system fails a recovery function is called. This function is not defined and can be anything.

Another family of geographical routing algorithms is the family that uses face routing in some variation, [7 8 9 10 11 12]. This type of algorithm guarantees delivery of the messages. However, it works correctly only in planar graphs, and might be inefficient in
route selections.

The first “Face routing” methods were proposed in 1999 in the articles ”Compass routing on geometric networks” [7] and ”Routing with guaranteed delivery in ad hoc wireless networks” [8]. These two similar methods pass the message along a sequence of adjacent faces until reaching the destination. This is done while looking for the edge that intersects the virtual source-target line. In the first method, the message is passed to the endpoint of this edge, where the routing on the next face begins. The second method also visits a sequence of faces, but it avoids crossing the source-target line, and switches to the next face just before that.

These early and important methods were improved later in many studies. We give two examples for “face routing” papers.

Brad Karp and H. T. Kung present GPSR [9], which makes greedy forwarding decisions using only information about a router’s immediate neighbors in the network topology. When a packet reaches a local minimum, the algorithm recovers by routing around the perimeter of the region. By keeping state information only about the local topology, GPSR scales better in the per-router state than shortest-path and ad hoc routing protocols as the number of network destinations increases.

Periodically, each node transmits a beacon to the broadcast MAC address, containing only its own identifier (e.g., IP address) and position. They encode position as two four-byte floating point quantities, for x and y coordinate values. The long-known right-hand rule for traversing a graph is used in face routing but does not always find routes when they exist (caused by non planar graph).

The Relative Neighborhood Graph (RNG) and Gabriel Graph (GG) are two planar graphs long-known in varied disciplines that can be used to remove crossing edges and keep the graphs connectivity. [21, 22]

GPSR combines greedy forwarding on the full network graph with perimeter forwarding
on the planarized (with GG or RNG) network graph where greedy forwarding is not possible.

In the papers implementation, they re-planarize the graph upon every acquisition of a new neighbor and every loss of a former neighbor. This is distinguishable by receipt of a beacon or data packet from a previously unknown neighbor, a timeout on a neighbors beacon, or MAC transmits failure indication.

A recent addition to the face routing family is GPVFR [12] by Ben Leong, Sayan Mitra, and Barbara Liskov. This paper improves routing efficiency by exploiting local face information. This paper proposes using more information about the planar graph. Having more information would lower the routing cost and lead to an algorithm with better routing performance in terms of both path and hop stretch. First they present a practical asynchronous distributed algorithm called ”Path Vector Exchange (PVEX)” . This algorithm propagates and maintains local face information efficiently as well as reacts to network membership changes.

Nodes in this paper hold information about all the faces they belong to. Every node has a number of faces exactly equal to its degree (in the planar subgraph), though in some cases, some of the faces will be repeated. The state associated with PVEX at node v has the following components: An indexed set of the faces adjacent to v, a list of successor nodes for face f, a list of predecessor nodes for face f, and sequence number for face f (an integer). On each face the successor and the predecessor are neighbors of v according to their position. They use a sequence number for the faces to distinguish between newer and older updates.

When greedy forwarding to an immediate neighbor fails, a node may find that it knows of another node along its planar faces that is nearer to the destination than itself. Then, the node will apply the OPVFR algorithm to choose a target node, creating ”virtual edges” for faces with incomplete information if necessary. Once a target node is chosen, it is recorded in the packet and the packet is switched to OPVFR mode and forwarded toward the target node. It is possible that this target node may be replaced by another if one that is nearer
to the destination than the recorded target node is found while the packet is forwarded in OPVFR mode. OPVFR forwarding is like greedy forwarding except that nodes have a longer horizon, and packets are restricted to forwarding on the planar faces (edges). Under both Greedy and OPVFR modes, a packet may end up at local minimum. If so, they resort to Face Routing. The choice of the direction to traverse the face is made based on the currently known set of path vectors instead of using an arbitrary right-hand rule like GPSR.

Bypassing voids can be done by different variations of face routing as we saw earlier. GDSTR overcomes the voids by routing based on a hall tree, which we found interesting and similar to our approach.

In the article by Ben Leong, Barbara Liskov, and Robert Morris [14], GDSTR is presented like previous geographic routing algorithms, forwarding packets using simple greedy forwarding whenever possible. It switches to forwarding on a spanning tree only to route packets around voids, and escape from a local minimum. It switches back to greedy forwarding as soon as it is feasible to do so. The reason GDSTR is able to guarantee the delivery of packets in a connected network is that the tree traversal forwarding mode is guaranteed to deliver the packet to any node in the network without greedy forwarding. In this paper the authors define and use hull trees. A hull tree is a spanning tree where each node has an associated convex hull that contains the locations of all its descendant nodes. The convex hull for a set of points is the convex polygon that contains all the points. To ensure that the convex hulls use only $O(1)$ storage instead of $O(n)$ storage, where $n$ is the network size, they limit the number of vertices for a convex hull to a maximum of $r$ points. In order to route packets on a hull tree, they forward a packet to child nodes that have a convex hull containing the destination. If none of the child nodes have convex hulls containing the destination, they know that the destination is not reachable down the tree, so they forward the packet up the tree. One complication that can arise is that the destination may lie in the intersection of
the convex hulls of two child nodes. In this case, they can still guarantee packet delivery
by a systematic depth first search on all the subtrees that have convex hulls containing the
destination.

Other existing geographical routing algorithms [15, 16, 17, 18, 19] use preprocessing of
the network topology with different methods in order to distribute virtual coordinates to
the nodes. This way they increase convexity and significantly reduce the probability of the
packet reaching a local minimum.

Rik Sarkar, Xiaotian Yin, Jie Gao, Feng Luo, and Xianfeng David [15] propose computing
a new embedding of the nodes in the plane such that greedy forwarding with the virtual
coordinates guarantees delivery. In particular, they extract a planar triangulation of the
network with non-triangular faces as holes. Virtual coordinates are computed with Ricci flow
such that all the non-triangular faces are mapped to perfect circles. Thus greedy forwarding
will never get stuck at a local minimum.

Computation of the virtual coordinates is performed in a preprocessing phase and can
be implemented by a local gossip-style computation. The method applies to unit disk graph
models and quasi-unit disk graph models.

In the article by Noa Arad and Yuval Shavitt [19], the authors propose computing virtual
coordinates to the nodes in the network as well. The basic idea is that nodes examine
whether they can be local minimums when routing a message to some direction in the
network according to their location and their neighbors’ locations. If the angle between two
adjacent node neighbors exceeds 180 degrees, then the node is necessarily concave for routing
in this direction. Their method indicates a concave node by elevating it. In an N-dimension
coordinate system, an N+1 dimension is added that indicates, if its coordinate is nonzero,
that the node is virtually repositioned. The rest of the N dimension coordinates are updated
as well to reflect the node’s connectivity.

The routing is performed mainly through the nodes that did not get virtual coordinates except when climbing to the destination and descending from the source. In this system greedy routing is not always successful and has network density limitations.
Chapter 3

Preliminaries

Defining the network model is a very important step in communication research as it defines the network element types and types of connections between them.

**Graph** or a Network is a representation of a set of routing elements also called nodes or vertices, where some pairs of the nodes are connected by links, also called edges. Nodes in the network have some memory and computing capabilities, and can communicate by sending messages over the links. The links in our case are assumed to be undirected, i.e., for any link connecting two nodes a message can be delivered in both directions. The nodes are the network components in charge of making the decisions of which neighbor should they deliver a message to in order for it to eventually get to its destination. They consider the message’s destination and the knowledge they have on the network stored in their memory. A graph is often denoted as $G(V, E)$ because it defines the set of vertices $V$ and the set of edges $E$.

**Node Degree** Two vertices connected by a link in the network are often called neighbors in the network. The set of neighbors of node $v$ is denoted as $N(v)$. Node Degree in a graph is the number of neighbors it has. Summing the degrees of all the nodes in a graph equals
exactly twice the number of edges in that graph.

Tree graphs are graphs with a special structure; we deal with this type of graph throughout this thesis.

**Tree Graph** in mathematics, more specifically graph theory, is a graph in which any two vertices are connected by exactly one simple path. In other words, a tree graph is a connected graph with no cycles. A tree graph with $N$ nodes has exactly $N - 1$ edges (Figure 3.1).

![Tree graph example](image)

Figure 3.1: Tree graph example

Both the Coding and Routing parts of this research rely on the special data structure called Quadtree and the hierarchy defined by the Quadtree graph.

**Quadtree** is a well known data structure that has many applications in image processing, computational geometry, and geometrics. The basic function of a Quadtree is to cover a planar region of interest with a square, than recursively partition it into smaller squares until each square contains only one vertex. This data structure was named a quadtree by Raphael Finkel and J.L. Bentley in 1974. A formal definition is given in 4.1.

**Quadtree Graph** is a tree graph in which each node represents a squared area, and an edge exists between any node to the four nodes representing its four quarters after partitioning.

24
This defines a tree hierarchy of squares where the node that represents the original square is the Quadtree graph root and each node representing a partitioned square is the parent of its four quarters nodes. A leaf square is a Quadtree square that is not partitioned. An example is shown in Fig. 1.1.

In the first part of this thesis we describe our new routing algorithm, prove its correctness, and show its efficiency by simulations.

**Routing Algorithm** Routing is the process of selecting paths in a network along which to send network traffic. The routing elements usually determine forwarding decisions based on the routing tables, which maintain a record of how to get to various network destinations. Thus, constructing routing tables, which are held in the nodes’ memory, is very important for efficient routing. In order to construct and maintain the routing tables, routers send each other messages containing information about the network. These messages are generally preferred to be minimal in size and quantity for many reasons such as saving network bandwidth and processing time. The size of the routing table held in the routers is often called the routers’ ”state”, which is preferred to be minimal as well.

The construction and management of the routing tables, in addition to the set of rules the routers look at when making the forwarding decisions, all combine the routing algorithm.

There are two main families of routing algorithms that include most of the existing algorithms. The first is the distance vector algorithms, and the second is the link state algorithms.

**Distance Vector Algorithms** In this approach every node assigns the cost of reaching each of the other nodes in the network via each of its neighbors. Nodes will send information from point A to point B via the path that results in the lowest total cost (i.e., the sum of the costs of the links between the nodes on the path). The algorithm operates in a very simple manner. When a node first starts, it only knows of its immediate neighbors, and the
direct cost involved in reaching them. (This information, the list of destinations, the total cost to each, and the next hop to send data to get there, makes up the routing table or distance table.) Each node, on a regular basis, sends to each neighbor its own current idea of the total cost to get to all the destinations it knows of. The neighboring node(s) examine this information, and compare it to what they already 'know'; anything that represents an improvement on what they already have, they insert in their own routing table(s). Over time, all the nodes in the network will discover the best next hop for all destinations, and the best total cost.

**Link-State Algorithms**  When applying link-state algorithms, each node uses as its fundamental data a map of the network in the form of a graph. To produce this, each node floods the entire network with information about what other nodes it can connect to, and each node then independently assembles this information into a map. Using this map, each router then independently determines the least-cost path from itself to every other node using a standard shortest paths algorithm such as Dijkstra’s algorithm. The result is a tree rooted at the current node such that the path through the tree from the root to any other node is the least-cost path to that node. This tree then serves to construct the routing table, which specifies the best next hop to get from the current node to any other node. This approach prefers increasing the information the nodes have on the network causing higher efficiency routing. The cost of obtaining this information is high management traffic both in size and quantity, in addition to complex computing in the routing nodes, which demand time and capabilities.

The routing algorithm we developed uses location information in order to route the messages; therefore it is a geographical routing algorithm.
**Geographic Routing**  Geographic routing (also called Geo routing or position-based routing) is a routing principle that relies on geographic position information. It is based on the idea that the source sends a message to the geographic location of the destination, instead of using the network address. Geographic routing requires that each node is able to determine its own location and that the source is aware of the location of the destination. With this information a message can be routed to the destination without knowledge of the network topology.

Most strategies rely on greedy forwarding and face routing. Greedy forwarding tries to bring the message closer to the destination in each step using only local information. In other words, each node forwards the message to the neighbor that is most suitable from a local point of view. The most suitable neighbor can be the one who minimizes the distance to the destination in each step (Greedy).

Greedy forwarding can lead into a dead end, where there is no neighbor closer to the destination. Then, face routing helps to recover from that situation and find a path to another node, where greedy forwarding can be resumed. A recovery strategy such as face routing is necessary to assure that a message can be delivered to the destination.

Many geographical routing algorithms, such as face routing, demand that the network graph be planar in order for them to work correctly, such as the following face routing system.

**Planar graph**  In graph theory, a graph G is said to be planar if it can be represented on a plane in such a fashion that the vertices are all distinct points, the edges are simple curves, and no two edges meet one another except at their terminals [23].

**Face Routing**  The key insight in face routing is the observation that a planar graph is composed of a set of well defined faces, each of which has nodes as its vertices. Suppose we
want to route a packet from a source node \(s\) to a destination node \(t\). The imaginary line \(st\) will then intersect a fixed number of faces as shown in Figure 3.2. By traversing the network along these faces, one will eventually reach the destination. A packet can traverse a face by using a simple right-hand rule. What this means is that when a node receives a packet on a given link, the packet is forwarded on the first link that is counterclockwise of the ingress link.

![Figure 3.2: Face routing using "right hand rule"](image)

In the second part of this thesis we define a new arithmetic coding system for nodes in a network, where the code words hold information about the geographical location of the nodes they represent.

**Arithmetic Coding** An arithmetic encoder takes a string of symbols as input and produces a rational number in the interval \([0,1)\) as output. As each symbol is processed, the encoder will restrict the output to a smaller interval. That way the code word is constructed gradually by adding postfix. In our case we normalize the coordinates of the nodes in the network to coordinates in a unit squared area, and the input would be the location of the nodes in the unit square. Since we use the arithmetic codes on binary computing nodes, the actual output of the encoder is the shortest sequence of bits representing the fractional part of both the \(X\) and \(Y\) coordinates (in the unit square) combined into one geographical address.
We use the Longest Matching Prefix measure in many calculations throughout this thesis.

**Longest Matching Prefix** A prefix is a part of a word attached to a beginning of a word. In our case a binary code word that represents the address of a node in a network. The longest matching prefix of two code words is the longest identical substring the two words have, starting at the left end bit of the words.
Part I

Geographical Quadtree Routing
Chapter 4

Problem Definition and Routing Idea

Our goal is to develop a distributed routing algorithm for a connected network $G(V,E)$ that enables any node to deliver a message to any other node in the network with guaranteed delivery. The nodes will "wake up" knowing only their own location, and the connections they have to other neighbors and communicate with their neighbors to obtain a geographical address and the needed routing information. As a geographical routing algorithm, this algorithm will take advantage of the geographical information the nodes have of their own, their neighbors, and the destination location, in order to route the messages in the network in such a way that the nodes can maintain a small routing state only. A node state in this routing algorithm is dependent on the degree of the node rather than the overall network size, which greatly reduces memory use and lookup time, and scales better. We avoid using global or even limited flooding as a routing technique, in addition to global broadcasts for sending routing information; in fact, all of the routing information is delivered in one hop broadcast.

The routing system we develop takes the advantage of the stateless hop efficient greedy routing, and the guaranteed delivery of tree based routing. It uses an infrastructure of MultiQuadtree which we define. A message is routed throughout the network in a greedy
manner until it gets to a local minimum, then switches to tree mode routing. The packet will return to greedy mode when passing the local minimum, meaning when the packet gets to a node that is closer to the destination than the last local minimum. Tree mode routing is made dynamically using edges that are not necessarily MultiQuadtree edges as long as the node we route the message to is closer to the destination on the tree. It is a novel concept of greedy routing based on the tree distance. Having the MultiQuadtree infrastructure lets us be sure such a route always exists.

4.1 Quadtree Geographical Address

The basic idea of a Quadtree is to cover a planar region of interest with a square, then recursively partition it into smaller squares until each square contains only one vertex. Consider a squared area in the plane, $S$, and a set of nodes (communication stations) $V$ distributed in $S$. We recursively partition $S$ into four smaller squares until each square contains only one vertex, i.e., empty sub-squares stop the partition. Let $Quadtree_S(V)$ be the unique Quadtree partitioning of $V$; in the following we assume $S$ is known and drop it from the notation. See, for example, Fig. 4.1 for a Quadtree we generated from a set of 17,168 weather stations around the world, from Mathematica 7.

The $Quadtree(V)$ partitioning can be used to give addresses to squares and nodes. An empty Quadtree address, $\emptyset$, represents the whole original square area, then recursively each of the four quarters will get the address of the partitioned square with an added suffix to represent which quarter it is. In the following manner, the bottom left quarter will get the ”00” suffix, the top left will get ”01”, the bottom right will get ”10” and the top right will get ”11”. A node’s address is the address of the smallest square that it is contained in after the described partitioning is done. For $v \in V$, let $Qadd(v)$ be the unique Quadtree address of node $v$ resulting from $Quadtree(V)$.
Figure 4.1: Quadtree of 17,168 weather stations around the world from Mathematica 7.

Figure 4.2: Quadtree address is built of suffixes added with each recursive partition of the square area according to the suffix table (top left). A Quadtree partition example for a set of nodes (right) and some address examples (bottom left)

This partitioning defines the tree hierarchy of squares: the original square is the Quadtree root and each partitioned square is the parent of its four quarters. A leaf square
is a Quadtree square that is not partitioned.

### 4.2 MultiQuadtree Network

The idea of a *Quadtree network* is to use the square hierarchy of $Quadtree(V)$ in order to create a routing hierarchy. This is done by first assigning *representatives* to each square in the Quadtree hierarchy and connecting representatives according to the Quadtree hierarchy. In a *MultiQuadtree network*, a node can represent one or more squares it is contained in, and a square can have one or more representative nodes. As we show, this increases load balancing, and allows finding short paths while keeping the hierarchy well defined. We now give notation and definitions to be used in this paper.

Let $\tau$ be a Quadtree address and $|\tau|$ be the length of the Quadtree address, defined as the number of *bit pairs* in $\tau$. A $k$-Qsquare is a squared area in $Quadtree(V)$, with an address $\tau$ such that $|\tau| = k$. Let $\alpha$ be a $k$-Qsquare, the *parent* of $\alpha$ is the unique $(k-1)$-Qsquare that contains $\alpha$. $\alpha$’s parent address is $\alpha$’s address removing the two-bit suffix. The *sub-Qsquares* of $\alpha$ are the four $(k+1)$-Qsquares contained in it. Their addresses are $\alpha$’s address with a two-bit suffix added. Let the *Ancestors* of $\alpha$ be all the $i$-Qsquares containing $\alpha$ such that $0 \leq i < k$. $\alpha$ is a *Populated Qsquare* iff it contains at least one node. $\alpha$ is a *Leaf Qsquare* iff it contains exactly one node.

For a Qsquare $\alpha$, let $RepSet(\alpha)$ denote the set of the nodes that are the representatives of $\alpha$. Recall that a representative of a Qsquare must belong to the Qsquare. We now define when a graph $G$ is considered a *MultiQuadtree*:

**Definition 1** A graph $G(V,E)$ is a MultiQuadtree network iff: There exists an assignment of representatives to all populated Qsquares in $Quadtree(V)$ s.t.

1. Any representative of populated $k$-Qsquare in $Quadtree(V)$, where $k > 0$, has a link (i.e., edge $e \in E$) to at least one representative of its parent Qsquare.
2. Any representative of populated $k$-Qsquare in $\text{Quadtree}(V)$, that is not a leaf $Q$-square, has links (i.e., edges $e \in E$) to at least one representative of each of its populated sub-$Q$-squares.

In the above definition we consider nodes as connected to themselves. A root of a Multi-Quadtree network is a representative of the $0$-Qsquare. Note that a graph that is a MultiQuadtree must have certain edges according to definition 1, but have no limitation on additional edges.

The following lemma is a consequence of the definition:

**Lemma 4.1** A MultiQuadtree network is connected.

**Proof.** Without loss of generality we assume node $v$ in a MultiQuadtree network with $|Qadd(v)| = i$. Let node $u$ be a $0$-Qsquare representative.

We prove lemma 4.1 by showing any $0$-Qsquare representative in a MultiQuadtree network is connected to all of the nodes in the network. According to property 2 in definition 1, Node $u$ is connected to a representative from all populated sub-Qsquare; therefore it is connected to a $1$-Qsquare representative that contains $v$. That step can be done recursively and show $u$ is connected to $w$, a representative of the $i-1$-Qsquare that contains $v$, ($w$ is $v$’s parent Qsquare). Node $v$ is the only representative of its leaf Qsquare that is an $i$-Qsquare. Node $w$ must be connected to $v$ as it is the only representative of a sub-Qsquare for $w$. Therefore $u$ is connected to $v$. ■

The definition of a MultiQuadtree network allows a node to represent any of the Qsquares it is contained in, that is any of its ancestor Qsquares. In addition to its leaf Qsquare which it always represents. Nodes hold the addresses of all the Qsquares they represent in the set $Tset(v)$. For $v \in V$, let $TSet(v)$ be the set of addresses of all the Qsquares $v$ represents, including its leaf Qsquare with the address $Qadd(v)$. Formally, for any Qsquare $\alpha$ such that $v \in RepSet(\alpha)$, $v$ will keep the Quadtree address $\overline{\alpha}$ in its $TSet(v)$. 35
Figure 4.3: MultiQuadtree network example: (A) $G(V, E)$ and $Quadtree(V)$. (B) The MultiQuadtree hierarchy. (C) $Q_{add}$ and $T_{set}$ table.

Fig. 4.3 illustrates the above definitions: (A) present graph $G(V, E)$, which is a MultiQuadtree network according to Definition 1 and the $Quadtree(V)$ partitioning. (B) shows the MultiQuadtree network hierarchy; arrows are stretched from Qsquare representatives to their populated sub-Qsquares representatives. (C) The $Q_{add}$ and $T_{set}$ of all the nodes, for example: nodes $d$ and $h$ represent the 0-Qsquare (they are both roots for this MultiQuadtree network), as they are both connected to representatives from all populated 1-Qsquare, which are $b, f, h$ for node $d$ and $b, g, h$ for node $h$. Node $h$, as a representative for Qsquare ”10”, is also connected to all of its populated sub-Qsquares representatives $i$ and $b$ itself. Node $f$ represents 11 as it is connected to the representatives of its populated sub-Qsquares $g$ and itself, in the same way node $g$ also represents 11. As one can see, a node can represent several different Qsquares it is contained in. For example, node $d$ represents three different Qsquares, their addresses are in $T_{set}(d)$. Moreover, a Qsquare can be represented by several
nodes. For example, the Qsquare with address 11 is represented by nodes \( f \) and \( g \), and the whole square is represented by the two roots \( d \) and \( h \).

One concern of a Quadtree Network is the need for "long" edges. We claim that the average length of edges in a Quadtree network is of the same order as the minimum spanning tree (MST). For example, the average edge length of the MST of \( n \) points located at the \( \sqrt{n} \times \sqrt{n} \) grid on the unit square is \( \frac{1}{\sqrt{n}} \), while the average edge length of a Quadtree Network of these points is \( \frac{2}{\sqrt{n}} \). We ran both MST and random Quadtree network construction on the points of Fig. 4.1; the average edge length in the MST came to 0.0018, while in the Quadtree Network it was 0.00503, which is only 2.78 times bigger.

### 4.3 Quadtree-Based Routing Algorithms

The main routing algorithm we offer is a combination of two routing modes, the first is greedy mode in which a node forwards a message to the neighbor that is physically closest to the destination (and closer than the node itself). This requires a way to calculate the physical distance based on the Quadtree addresses, which are the only information a node has on the location of its neighbors and of the destination. We call this function \( G_{dist} \). The second routing mode is called Quadtree mode. In this mode a node routes a message to the neighbor that is closest in the MultiQuadtree hierarchy to the destination. We call the function of calculating this type of distance \( MQ_{dist} \).

Let \( w, d \) be two nodes in \( V \). Let \( G_{dist}(w, d) \) be the Greedy distance between nodes \( w \) and \( d \).

- \( G_{dist}(w, d) \) is the Euclidean distance between the bottom left corners of the squared area represented by \( Qadd(w) \) and \( Qadd(d) \), the Quadtree addresses of the nodes.

Now since leaf Qsquares are disjoint, clearly:

**Lemma 4.2** \( G_{dist}(w, d) \in \mathbb{R}^+ \) and \( G_{dist}(w, d) = 0 \) iff \( w = d \).
Proof. According to definition, $G_{\text{dist}}(w, d)$ is the Euclidean distance between leaf Qsquares $v$ and $w$, bottom left corners. According to Lemma 10.2 the leaf Qsquares of $v$ and $w$ where $v \neq w$ are disjoint squares; therefore $G_{\text{dist}}(w, d) > 0$ for $w \neq d$, and $G_{\text{dist}}(w, d) = 0$ for $w = d$. ■

In order to define $MQ_{\text{dist}}$ let $lmp(x, y)$ to be the longest, even length, matching prefix of $x$ and $y$, where $x, y$ are two Quadtree addresses. A closer look at $lmp(x, y)$ shows us that, by definition, it is the Quadtree address of the smallest common ancestor Qsquare of the squares represented by addresses $x$ and $y$.

Let $MQ_{\text{dist}}(w, \alpha)$ be the MultiQuadtree network distance between node $w$ and Qsquare $\alpha$ with address $\overline{\alpha}$.

- $MQ_{\text{dist}}(w, \overline{\alpha})$ is calculated in the following way:

$$\min_{w \in TSet(w)} (|w| + |\overline{\alpha}| - 2 \times lmp(w, \overline{\alpha}))$$

For example, the $MQ_{\text{dist}}$ between $e$ and $\overline{k}$ in Fig. 4.3, where $\overline{k} = \text{Qadd}(k)$, is:

$$MQ_{\text{dist}}(e, \overline{k}) = \min(|0100| + |011110| - 2 \times lmp(0100, 011110), |01| + |011110| - 2 \times lmp(01, 011110)) = \min(2 + 3 - 2 \times 1, 1 + 3 - 2 \times 1) = 2.$$

Lemma 4.3 $MQ_{\text{dist}}(w, \overline{\alpha}) \in \mathbb{N}$ and $MQ_{\text{dist}}(w, \overline{\alpha}) = 0$ iff $w \in \text{RepSet}(\alpha)$.

Proof. For any $\overline{\alpha}, \overline{\gamma}$, the sizes $|\overline{\alpha}|, |\overline{\gamma}|, |lmp(\overline{\alpha}, \overline{\gamma})| \in \mathbb{N}$ by definition, and $|lmp(\overline{\alpha}, \overline{\gamma})| \leq \min(|\overline{\alpha}|, |\overline{\gamma}|)$ by definition. Therefore $MQ_{\text{dist}}(w, \overline{\alpha}) \in \mathbb{N}$.

Assume by contradiction $w \notin \text{RepSet}(\alpha)$ and $MQ_{\text{dist}}(w, \overline{\alpha}) = 0$, meaning there is an address $\overline{w} \in TSet(w)$ such that (*)$|\overline{\alpha}| + |\overline{w}| - 2 \times |lmp(\overline{\alpha}, \overline{w})| = 0$. We know $|lmp(\overline{\alpha}, \overline{w})| \leq \min(|\overline{\alpha}|, |\overline{w}|)$ $\Rightarrow \overline{\alpha} = \overline{w}$. We also know $\overline{w} \in TSet(w)$. So $\overline{\alpha} \in TSet(w)$ Therefore $w \in \text{RepSet}(\alpha)$ Which is in contradiction to the assumption.

This showed $MQ_{\text{dist}}(w, \overline{\alpha}) = 0 \Rightarrow w \in \text{RepSet}(\alpha)$.
Now showing $w \in \text{RepSet}(\alpha) \Rightarrow MQ_{\text{dist}}(w, \overline{\alpha}) = 0$ is simply by calculation. Assume $w \in \text{RepSet}(\alpha) \Rightarrow \overline{\alpha} \in \text{TSet}(w)$, consider $\overline{\alpha} = \overline{w}$, $MQ_{\text{dist}}(w, \overline{\alpha}) = |\overline{\alpha}| + |\overline{\alpha}| - 2 \times |\overline{\alpha}| = 0$. We also know $MQ_{\text{dist}}(w, \overline{\alpha}) \leq 0$, Therefore $MQ_{\text{dist}}(w, \overline{\alpha}) = 0$.

**Lemma 4.4** $MQ_{\text{dist}}(w, \overline{d}) = 0$ where $\overline{d} = Qadd(d)$ iff $w = d$.

Again the distance is zero only when the two are the same:

**Proof.** $MQ_{\text{dist}}(w, \overline{d}) = 0 \Rightarrow$ according to lemma 4.3 $\overline{d} \in \text{TSet}(w)$. $\overline{d} = Qadd(d)$ shows the Qsquare with address $\overline{d}$ is a leaf Qsquare. But the only node representing a leaf Qsquare is the node in it. $\Rightarrow w = d$. ■

Lemma 4.2 and 4.4 present two properties that will allow us to route messages in the network with guaranteed delivery.
Chapter 5

GQG Routing Algorithm

In this chapter we present two routing algorithms that are based on the quadtree addressing system and on the two distance types defined in section 4.3. First we present the Q routing algorithm, which routs the messages in a greedy manner using the $MQ_{\text{dist}}$ distance, meaning the message is forwarded to the neighbor that is closest on the MultiQuadtree to the destination. Secondly we present GQG, which forwards the messages in a geographical greedy manner, and uses the Q routing algorithm for local minimum recovery.

5.1 Quadtree Routing with Shortcuts

Quadtree mode routing works using edges that are not necessarily the tree edges, as long as the node we route the message to is closer to the destination on the tree. This is a novel concept of greedy like routing that is based on the tree distance we defined earlier - $MQ_{\text{dist}}$. We calculate the tree distance from a node to the Quadtree address of the target based on the $TSet$ of the node, which defines the locations of the node in the MultiQuadtree network hierarchy. This way a node looks at all its neighbors, and may use any link it has when routing in Quadtree mode rather than just looking at its parent and sub-Qsquares in
the hierarchy. This property leads to a better hop efficiency and better load balancing, as simulations show.

Let \( N(v) \) denote the neighbors of \( v \), i.e., all nodes that \( v \) is connected to. The Quadtree mode routing is defined in algorithm 1.

**Algorithm 1 Q: Quadtree routing**

A source node \( s \) is sending a message \( m \) to a target node \( t \) with \( \bar{t} = Qadd(t) \). Each node \( v \neq t \) that receives does the following:

1. find neighbor \( u^* = \arg\min_{u \in N(v)} (MQ_{dist}(u, \bar{t})) \)
2. If \( MQ_{dist}(u^*, \bar{t}) < MQ_{dist}(v, \bar{t}) \)
   (a) forward the message to \( u^* \).
3. else:
   (a) destination unreachable - terminate the message.

The following theorem states the correctness of algorithm 1 and bounds its performance.

**Theorem 2** Let \( u \) and \( d \) be nodes in a MultiQuadtree network where \( \bar{d} = Qadd(d) \). Routing a message \( m \) from \( u \) to \( d \) with Algorithm 1 has guaranteed delivery. Moreover, \( MQ_{dist}(u, \bar{d}) \) is an upper bound of the number of hops in the route.

**Proof.** Recall that a MultiQuadtree network is connected and consider a message \( m \), currently at node \( v \), with destination node \( d \) where \( Qadd(d) = \bar{d} \). We will prove that a node \( v \neq d \) always has neighbor \( w \) s.t. \( MQ_{dist}(w, \bar{d}) < MQ_{dist}(v, \bar{d}) \), so \( m \) is forwarded to a node with a lower \( MQ_{dist} \) to \( \bar{d} \) in every hop. Following Lemma 4.4 such an algorithm will only stop when \( v = d \). Now, assume \( v \neq d \), let \( \alpha \) be a Qsquare that is represented by \( v \) and has
the address:

\[ \overline{\alpha} = \overline{\gamma}^* = \arg\min_{\overline{y} \in TSet(v)} |\overline{y}| + |\overline{d}| - 2 \times \text{lmp}(\overline{y}, \overline{d}) \]

Now, there are two options:

**Qsquare \( \alpha \) is an ancestor of the leaf Qsquare of \( d \):** Since \( v \) is a representative of \( \alpha \) it must be connected to \( w \), a representative of the sub-Qsquare of \( \alpha \) that contains \( d \). Therefore, there is an address \( \overline{x} \in TSet(w) \) such that \( |\text{lmp}(\overline{x}, \overline{d})| = |\text{lmp}(\overline{\alpha}, \overline{d})| + 1 \) and \( |\overline{x}| = |\overline{\alpha}| + 1 \) so, \( MQ_{\text{dist}}(v, \overline{d}) = |\overline{\alpha}| + |\overline{d}| - 2 \times |\text{lmp}(\overline{\alpha}, \overline{d})| \) and \( MQ_{\text{dist}}(w, \overline{d}) = (|\overline{\alpha}| + 1) + |\overline{d}| - 2 \times (|\text{lmp}(\overline{\alpha}, \overline{d})| + 1) \), which implies \( MQ_{\text{dist}}(w, \overline{d}) < MQ_{\text{dist}}(v, \overline{d}) \).

**Qsquare \( \alpha \) is not an ancestor of the leaf Qsquare of \( d \):** Since \( v \) is a representative of \( \alpha \) it must be connected to node \( w \), which is the representative of the parent Qsquare of \( \alpha \). Therefore there is an address \( \overline{x} \in TSet(w) \) with \( |\text{lmp}(\overline{x}, \overline{d})| = |\text{lmp}(\overline{\alpha}, \overline{d})| \) and \( |\overline{x}| = |\overline{\alpha}| - 1 \) so, \( MQ_{\text{dist}}(v, \overline{d}) = |\overline{\alpha}| + |\overline{d}| - 2 \times |\text{lmp}(\overline{\alpha}, \overline{d})| \) and \( MQ_{\text{dist}}(w, \overline{d}) = (|\overline{\alpha}| - 1) + |\overline{d}| - 2 \times |\text{lmp}(\overline{\alpha}, \overline{d})| \) which implies \( MQ_{\text{dist}}(w, \overline{d}) < MQ_{\text{dist}}(v, \overline{d}) \). □

### 5.2 GQG: Greedy-Quadtree-Greedy Routing Algorithm

We now define the Greedy-Quadtree-Greedy routing algorithm for routing a message in a graph \( G(V, E) \), which is a MultiQuadtree network. A source node \( s \) is sending a message \( m \) to a target node \( t \) with \( \overline{t} = Qadd(t) \). The message \( m \) holds the following four fields: (1) \textit{m.source}, (2) \textit{m.target}, (3) \textit{m.mode}, and (4) \textit{m.localmin}. Field (1) is used to identify the message source. (2) is used to identify the target Quadtree address. (3) is used to identify the routing mode, which can be \{tree/greedy\}. (4) is used to store the \( G_{\text{dist}} \) between the target and the last local minimum node, in order to determine whether it is possible to change the routing mode from \textit{tree} mode back to \textit{greedy}. The message is initialized with the following parameters: \( m\textit{.source} = Qadd(s) \), \( m\textit{.target} = Qadd(t) \), \( m\textit{.mode} = \text{greedy} \) and \( m\textit{.localmin} = G_{\text{dist}}(s, t) \). \textit{GQG} is presented in Algorithm 2.
Algorithm 2 GQG: Greedy-Quadtree-Greedy routing

The algorithm starts at $s$ and defines the behavior of any node $v \neq t$ that receives $m$ destined to node $t$: (otherwise $v$ is the target and the algorithm stops).

1. Check $m.mode$:
   
   (a) greedy: perform step 2.
   
   (b) tree: perform step 3.

2. Greedy mode:
   
   (a) find neighbor $u^* = \arg\min_{u \in N(v)} G_{\text{dist}}(u, t)$
   
   (b) if $G_{\text{dist}}(u^*, t) < G_{\text{dist}}(v, t)$
       
       i. forward the message to $u^*$.
   
   (c) else:
       
       i. $m.localmin \leftarrow G_{\text{dist}}(v, t)$.
       
       ii. change $m.mode$ to tree.
       
       iii. perform step 4.

3. Mode examine:
   
   (a) If $G_{\text{dist}}(v, t) < m.localmin$.
       
       i. change $m.mode$ to greedy
       
       ii. perform step 2.
   
   (b) else:
       
       i. perform step 4.

4. Quadtree mode:
   
   (a) Follow Algorithm 1.
5.3 Algorithm Correctness

Theorem 4 states the correctness of the GQG algorithm.

Definition 3 Let Routing Step be a sequence of one or more hops starting at any node \( v \in V \setminus d \) and ending at node \( u \in V \) such that \( m.mode = \text{greedy} \) only at \( v \) and \( u \).

Lemma 5.1 All routing steps end in a finite number of hops.

Proof. There are two types of routing steps: The first is when the message is delivered from node \( v \) to node \( u \) in greedy mode (step 2.b in the algorithm). In this case the routing step ends after one hop, as \( m.mode = \text{greedy} \) at \( v \) and \( u \). The second is when the message is delivered from node \( v \) to node \( u \) in tree mode (step 4.b in the algorithm). According to lemma 2 routing in tree mode has a guaranteed delivery; therefore tree mode will always return to greedy mode, which in the best case will be at the target node \( t \). Tree mode routing has the property of reducing the \( MQ_{dist} \) at every hop; therefore it takes at most \( N-1 \) hops to switch back to greedy mode.

Theorem 4 GQG has guaranteed delivery in finite MultiQuadtree networks.

Proof. Consider routing a message \( m \) towards target \( t \) in a network with \( N \) nodes using GQG routing algorithm. Routing step has the property of \( G_{dist}(u,t) < G_{dist}(v,t) \); therefore there are maximum \( N-1 \) Routing Steps between any two nodes in the network. According to lemma 5.1 all routing steps end in a finite number of hops, and are bounded by \( N-1 \) hops. Therefore GQG has guaranteed delivery bounded by \( N^2 \) hops.
5.4 Routing Example

Figure 5.1: Routing example - MultiQuadtree infrastructure is marked in solid lines and additional edges in dashed lines.

The network shown in 5.1 has an infrastructure of MultiQuadtree marked in solid lines and additional edges in dashed lines. We show an example of routing a message from node 1 to node 9 with the GQG algorithm. $Q_{add}(9)$ is: 00 10.

- Node 1 sends the message in greedy mode to node 2 then to 4, then to node 5 and to node 6.

- Node 6 is a local minimum as it looks for a neighbor $u^*$ such that $u^* = \arg\min_{u \in N(v)} G_{dist}(u, 9)$, finds Node 5 but $G_{dist}(5, 9) > G_{dist}(6, 9)$. So the message mode is changed to tree mode and the $G_{dist}(6, 9)$ is set in the message at the $m.localmin$ parameter.

- Node 6 then looks for a neighbor $u^*$ such that $u^* = \arg\min_{u \in N(v)} MQ_{dist}(u, "0010")$ in order to route the message to it.
Qadd(6) is: 10 00 00. $MQ_{\text{dist}}(6,"0010")$ is $\|100000\|+\|0010\| - 2 \times |lmp("100000","0010")|$
\[
= 3 + 2 - 0 \times 2 = 5 
\]

Qadd(5) is: 10 00 01. and 10 00 $\in Tset(5)$. $MQ_{\text{dist}}(5,"0010")$ is calculated with this address because we use the minimum and it equals: $2 + 2 - 0 \times 2 = 4$. $MQ_{\text{dist}}(5,"0010") < MQ_{\text{dist}}(6,"0010")$ so message will be routed to it.

- Node 5 receives a message with tree mode, so it first checks whether it can switch to greedy mode (Step 3), but it can’t. Then it looks for a neighbor $u^*$ such that $u^* = \text{argmin}_{u \in N(v)} MQ_{\text{dist}}(u,"0010")$ in order to route the message to it.

- We already saw $MQ_{\text{dist}}(6,"0010") = 5, MQ_{\text{dist}}(5,"0010") = 4$.
- $MQ_{\text{dist}}(3,"0010") = 5$ in the same way as $MQ_{\text{dist}}(6,"0010")$.
- Qadd(4) is: 10 01 and 10 $\in Tset(4)$. $MQ_{\text{dist}}(4,"0010")$ is calculated with this address because we use the minimum and it equals: $1+2-0 \times 2 = 3$. $MQ_{\text{dist}}(4,"0010") < MQ_{\text{dist}}(5,"0010")$ so the message can be routed to it.

- Qadd(8) is: 00 11 01 and has the addresses 00, 0011 $\in Tset(8)$. $MQ_{\text{dist}}(8,"0010")$ is calculated with the last address because we use the minimum and it equals: $1 + 2 - 1 \times 2 = 1$ and the message will be routed to it because it is closer on the tree even though it is not the father of node 5 on the tree and in fact is located in a different branch.

- Node 8 receives a message with tree mode, and delivers it to node 9, which has $MQ_{\text{dist} 0}$ to itself.
Chapter 6

MQD MultiQuadtree Network Discovery

Both Quadtree and GQG routing algorithms rely on the fact that nodes are familiar with their own and their neighbors $TSete$, which of course must be configured in some manner. We offer $MQD$ - MultiQuadtree Discovery algorithm for giving the additional Qsquares representation addresses based on the network topology. This algorithm is distributed and uses local information. The $MQD$ algorithm is initiated once all the nodes in the network have their own and their neighbors Quadtree addresses. It does not change the topology, therefore can be completed iff there is such a MultiQuadtree induced in the network. After running the algorithm all nodes in the network are familiar with their $TSets$, and notify their neighbors. A requirement for this algorithms correctness is that all Qsquares are intra-connected, which is always true under the assumption there is an induced MultiQuadtree in the network.
6.1 Algorithm Idea

The MQD algorithm is built from two main phases:

Phase I: Works bottom-up from the MultiQuadtree network leaves towards its root(s). This phase insures that in order for \( v \) to become a representative candidate of Qsquare \( \alpha \), the following two conditions must be satisfied: 1. \( v \) is contained in \( \alpha \); and 2. \( v \) is connected to a representative candidate of each of the populated sub-Qsquares of \( \alpha \) (i.e., condition 2 in definition 1). Messages in this phase are sent only by candidates; initially all nodes are candidates for their leaf Qsquares, but their quantity decreases as the algorithm climbs the Quadtree hierarchy.

Phase II: Works its way from the MultiQuadtree network root(s) towards the leaves. The purpose of this phase is that a representative candidate for a Qsquare \( \alpha \) will become a representative for \( \alpha \) if and only if it is connected to a representative of \( \alpha \)'s parent Qsquare (i.e., condition 1 in definition 1), unless \( \alpha \) is the root Qsquare and becomes a representative without having a parent.

The selection of a representative is complex since we want a distributed and local algorithm, i.e., nodes only know about their direct neighbors and can only send messages to them. The complexity arises when, in order for a node to become a representative candidate for a Qsquare \( \alpha \), it has to know which of \( \alpha \)'s sub-Qsquares are populated. The problem is that a node that populates a certain sub-Qsquare can still be several hops away from a potential representative. We overcome this challenge by defining a data structure called population map (Pmap).

6.2 Population Maps

A population map of a node \( v \) is a matrix of bits denoted by \( \text{Pmap}(v) \), which keeps the knowledge \( v \) has about population in the square. The size of the map is determined by the
address length of \( v \). Node \( v \) uses its population map in the first phase of the MQD algorithm to inform its neighbors of the existing populated Qsquares it knows of. Next, Pmaps are aggregated and combined in such a way that when a node decides whether it can become a representative candidate for Qsquare \( \alpha \), it already knows which of \( \alpha \)'s sub-Qsquares are populated. Therefore it only needs to check if, for each of \( \alpha \)'s populated sub-Qsquares, it received a Phase I message from a neighbor that is a representative candidate of that sub-Qsquare.

To define the Pmap data structure we need to define the \textit{Sibling} relation: two Qsquares, \( \alpha \) and \( \beta \), are \textbf{k-Sibling Qsquares} if \( \text{lmp}(\alpha, \beta) = k \). Similarly, two nodes, \( u \) and \( w \) are \textbf{k-Sibling nodes} if \( \text{lmp}(\text{Qadd}(u), \text{Qadd}(w)) = k \). A node \( u \) is a \textbf{k-Sibling} of a Qsquare \( \alpha \) if \( \text{lmp}(\alpha, \text{Qadd}(u)) = k \).

Let the function \( \text{prefAdd}(k, \text{Qadd}(v)) \) return the \( 2 \times k \) bit prefix of the address \( \text{Qadd}(v) \).

The population map for node \( v \), Pmap\((v)\), is a matrix of bits of size \( 4 \times |\text{Qadd}(v)| \). Rows present the 4 sub-Qsquares and are numbered 0 to 3, columns present Qsquare level and are numbered from 0 to \( |\text{Qadd}(v)| - 1 \). Node \( v \) will have '1' in its population map at cell \([i, j]\) iff \( v \) knows that its \( j \)-Sibling Qsquare with index \( i \) is populated and '0' otherwise.

For example, in Figure 6.1(A) we see the initial population map of node \( d \) in the network shown in Fig. 4.3. Node \( d \) has \( \text{Qadd}(d) = 01 \ 11 \ 00 \), it knows that the \( j \)-Qsquares \((j = 0, 1, 2)\) it is contained in are populated; therefore cells \([1, 0], [3, 1], [0, 2]\) in Pmap\((d)\) are set to '1'. Its neighbor \( e \) has \( \text{Qadd} 01 \ 11 \ 10 \), and is a 2-sibling of \( d \); therefore \( d \) knows that the 2-Qsquare 01 11 with index 2 (i.e., 10) is populated and cell \([2, 2]\) is set to '1'. In the same way cell \([3, 2]\) set to '1' because of its neighbor \( c \). Neighbor \( b \) with address 01 00, is 1-sibling; therefore the cell is \([0, 1]\) set to '1'. And the 0-sibling neighbors \( h, f \) with addresses 10 01, 11 00 set cells \([1, 0], [0, 0]\).

In Figure 6.1(B) we can see the initial population map of node \( b \) in the network of Fig. 4.3.

When a node \( v \) receives from a neighbor \( u \) its Pmap\((u)\) and its quadtree address \( \text{Qadd}(u) \), it updates its own Pmap\((v)\) with what \( u \) knows according to the function LearnMap\(_v\)(\( u \)):
Figure 6.1: Example of population maps of nodes from the network shown in Fig. 4.3 (A) The initial population map of node \( d \), (B) The initial population map of node \( b \), and (C) The population map of node \( d \) after receiving and learning the population map of node \( b \) after executing \( \text{LearnMap}_d(b) \).

\[
\text{LearnMap}_v(u) := \begin{cases} 
\text{Pmap}(v)[i,j] & j > \text{lmp}(Q\text{add}(v), Q\text{add}(u)), i \in \{0, 1, 2, 3\} \\
\text{Pmap}(v)[i,j] \lor \text{Pmap}(u)[i,j] & j \leq \text{lmp}(Q\text{add}(v), Q\text{add}(u)), i \in \{0, 1, 2, 3\}
\end{cases}
\]

Figure 6.1(C) presents the population map of node \( d \) after receiving and learning the population map of node \( b \). Bit \([2, 1]\) is now set to one since node \( d \) received the information about the existence of node \( a \) from the population map of node \( b \). Consequently, node \( d \) now knows it cannot represent the \( \text{Qsquare } 01 \), because it has no neighbor in sub-\( \text{Qsquare } 01 10 \) and in particular no neighbor that is a representative candidate for the sub-\( \text{Qsquare } 0101 \).

There are two types of messages sent by node \( v \in V \) in MQD algorithm:

**RepReq.** Representing Request for t-\( \text{Qsquare } \alpha \): Node \( v \) sends this message to all of its neighbors iff it becomes \( \alpha \)'s representative candidate. This message holds the following fields: (1) source field = \( Q\text{add}(v) \), (2) t field = \(|\alpha|\), and (3) Pmap field containing current \( \text{Pmap}(v) \).

**RepCon.** Representing Confirm for t-\( \text{Qsquare } \alpha \): Node \( v \) sends this message to all of its neighbors once it becomes a representative of \( \text{Qsquare } \alpha \). This message holds the following fields (1) source field = \( Q\text{add}(v) \), and (2) t field = \(|\alpha|\).
Algorithm 3 MultiQuadtree Discovery

*Input*: A network with induced MultiQuadtree. *Output*: All nodes in the V initialize their TSets according to the MultiQuadtree definition. A node v behaves like the following:

1. Initialization:
   
   (a) Build initial population map, Pmap(v).
   
   (b) Become representative candidate for the leaf Qsquare.
   
   (c) Broadcast RepReq for |Quadd(v)| − Qsquare containing Qadd(v) and Pmap(v).

2. On receiving RepReq for a t-Qsquare from node u:
   
   (a) Pmap(v) = LearnMapv(u).
   
   (b) If received RepReq, for all populated t-sibling Qsquares where t > 1:
       
       i. become representative candidate for t-1-Qsquare
       
       ii. repAdd = prefAdd(t-1, Qadd(v))
       
       iii. insert repAdd to the “waiting for confirm” list.
       
       iv. broadcast RepReq for t-1-Qsquare containing repAdd and Pmap(v).
   
   (c) If received RepReq, from all populated t-sibling Qsquares where t = 1:
       
       i. become a MultiQuadtree root.
       
       ii. insert the address φ to Tset(v).
       
       iii. broadcast RepCon for 1-Qsquare with φ.

3. On receiving RepCon for t-Qsquare from a k-sibling node such that k ≥ t:
   
   (a) If waiting for confirm list contains an address repAdd such that |repAdd| = t:
       
       i. insert repAdd to the TSet(v).
       
       ii. broadcast RepCon for Qsquare t+1 with repAdd.
6.3 Algorithm Correctness

We prove MQDs correctness by showing the following:

1. Node $v$ becomes a representative candidate for $k$-Qsquare $\alpha$ iff $v$ is directly connected to at least one representative candidate from each of $\alpha$’s populated sub-Qsquares and has learned their Pmaps.

2. Node $v$ becomes a representative for $k$-Qsquare $\alpha$ iff $v$ is a representative candidate of $\alpha$ and for any $k > 0$ is directly connected to at least one representative of the $k$-1-Qsquare it is in.

Conditions 1 and 2 prevent a node from becoming a false representative, in addition for making sure that any node that can represent a Qsquare will do so. Therefore the maximal induced MultiQuadtree will be discovered.

Let $\beta$ be a Qsquare in $G$. Let $\text{MaxRepSet}(\beta)$ be all of the nodes contained in Qsquare $\beta$, which according to MultiQuadtree definition in 1 can represent $\beta$.

**Lemma 6.1** Node $v$ becomes a representative candidate for $k$-Qsquare $\alpha$ iff $v$ is connected to at least one representative from each of $\alpha$’s populated sub-Qsquares and has learned their Pmaps.

**Proof.** By induction: Let $G(V,E)$ be a MultiQuadtree network.

**Base case:** Let $\alpha$ be a populated Qsquare where all of its sub-Qsquares are leaf Qsquares. $G(V,E)$ is a MultiQuadtree network; therefore there exists a node $u \in V$ that is connected with an edge to all of the other nodes in $\alpha$. After the initialization (step 1.c in algorithm 3), node $v$ will receive the $\text{repReq}$ and the initial Pmaps from all other nodes in $\alpha$ as it is connected to them; therefore $v$ becomes a representative candidate for $\alpha$. The rest of the nodes in $\alpha$ receive $v$’s RepReq and initial Pmap that contain the information about
the populated sub-Qsquares of $\alpha$; therefore they will all become representative candidates of $\alpha$ iff they are connected to a node from each of them.

**Induction hypothesis:** Node $v$ becomes a representative candidate for k-Qsquare $\alpha$ iff $v$ is connected to at least one representative from each of $\alpha$’s populated sub-Qsquares and has learned their Pmaps.

**Induction step:** We will prove the correctness of the hypothesis for k-1-Qsquare. Let $\beta$ be a k-1-Qsquare in G, and $u \in MaxRepSet(\beta)$ as shown in Figure 6.2. The initial Pmap($u$) contains information about all of the populated sub-Qsquares of $\beta$. Let $\alpha$ be the sub-Qsquare of $\beta$ that contains $u$. Qsquare $\alpha$ has one or more representative candidates, which according to the assumption have learned Pmap($u$) and know about all other populated sub-Qsquares of $\beta$. This information will be contained in their Pmaps, and in the RepReq they send. Without loss of generality, we assume there are three other populated sub-Qsquares of $\beta$ denoted by $\delta, \psi, \zeta$. In each of the Qsquares $\delta, \psi, \zeta$, there must be a node that is connected to $u$, and got the initial Pmap($u$). Therefore, according to the assumption, $\delta, \psi, \zeta$ representative candidates learned Pmap($u$) and know about all other populated sub-Qsquares of $\beta$. A node $w$ in $\beta$ that receives RepReq from any of $\alpha, \delta, \psi, \zeta$ representative candidates will learn the information propagated from the initial Pmap($u$) about the populated sub-Qsquares of $\beta$; therefore it will become a representative candidate of $\beta$, iff it is directly connected to at least one representative candidate from each populated t-sibling Qsquare $\alpha, \delta, \psi, \zeta$.

![Figure 6.2: MultiQuadtree network, MQD algorithm correctness proof](image)

Figure 6.2: MultiQuadtree network, MQD algorithm correctness proof
**Lemma 6.2** Node $v$ becomes a representative for $k$-Qsquare $\alpha$ iff $v$ is a representative candidate of $\alpha$ and for any $k > 0$ is directly connected to at least one representative of the $k$-1-Qsquare it is in.

**Proof.** According to step 3 in the algorithm, only $t$-Qsquare representative candidates that receive a Representing Confirm message from a $k$-sibling node that is a representative of the parent Qsquare become representatives; this message is broadcast within one hop range, therefore they are directly connected. ■

**Theorem 5** After the completion of algorithm MQD, all nodes in $V$ initialize their TSets according to the MultiQuadtree definition.

**Proof.** According to Lemma 6.1 the MultiQuadtree found by the MQD algorithm follows condition 2 in definition and according to Lemma 6.2 the MultiQuadtree found by the MQD algorithm follows condition 1 in definition. Therefore the Tsets initialized by the nodes, after running MQD algorithm, follow the MultiQuadtree definition. ■

**Definition 6** Maximal MultiQuadtree($V$):

Let Maximal MultiQuadtree($V$) be a MultiQuadtree($V$) where each node that can be a $qsquare\alpha$ representative will be $qsquare\alpha$ representative.

**Theorem 7** After running the algorithm we obtain a Maximal MultiQuadtree according to definition.

**Proof.** According to step 2 in the MQD algorithm, any node that can become representative candidate for $qsquare\alpha$ will be. And according to step 3, any node that is a representative candidate for $qsquare\alpha$ and can be a representative will be. ■
Chapter 7

QAD Quadtree Address Distribution Algorithm

We offer a distributed algorithm for obtaining the address $\text{Qadd}(v)$ for all $v \in V$ based on the geographical location of the nodes in the unit square, called $QAD$ - Quadtree Address Distribution Algorithm. This algorithm is initiated once all the nodes in the network have their own coordinates at the unit square, and are able to communicate with their neighbors. After running the algorithm every node $v \in V$ in the network is familiar with its $\text{Qadd}(v)$, and notifies its neighbors. A requirement for this algorithms correctness is that all Qsquares are intra-connected.

Let $k-$Qadd($v$) be the address of the $k$-Qsquare that contains $v$. Let the function $\text{ATQ}(\overline{x})$ (Address To Qsquare) return the Qsquare with address $\overline{x}$.

The idea behind this algorithm is very simple. Nodes in the network initialize their Quadtree addresses to be an empty address and send it to their neighbors. Then when a node $v$ receives a message from a node $u \in N(v)$ containing $u$’s current address, $v$ checks whether its own current address equals or is a prefix of $u$’s address. In case it does, $v$ will add a two-bit suffix to its address according to its location and send its address again. Each
node stops when its address does not prefix any of its neighbors addresses.

While running the algorithm every node holds a list of nodes $Done(v)$ of its neighbor nodes that already have addresses that cannot effect its own. Let $Done(v) \subseteq N(v)$ be a list of the neighbor nodes where $l_{mp}(k - Qadd(v), Qadd(u)) < min(k, |Qadd(u)|)$.

Algorithm 4 QAD

*Input:* A graph $G(V,E)$ where all nodes know their self-coordinates at the unit square.

*Output:* Every node $v \in V$ in the network is familiar with its Qadd($v$), and notifies its neighbors.

1. Initialization:
   
   (a) $k = 0$

   (b) Broadcast $k$-Qadd($v$)

2. When receiving an address Qadd($u$) from a neighbor $u \in N(v)$:
   
   (a) if $l_{mp}(k - Qadd(v), Qadd(u)) < min(k, |Qadd(u)|)$
      
      i. $u \rightarrow Done(v)$
      
      ii. if $Done(v) = N(v)$
          
          A. Finish

   (b) if $l_{mp}(k - Qadd(v), Qadd(u)) = k$:
      
      i. $k = k + 1$
      
      ii. Broadcast $k$-Qadd($v$)
7.1 Algorithm Correctness

Claim 8 It is easy to show from definitions of Quadtree and k-Qsquare that
\( \text{lmp}(k-\text{Qadd}(v), \text{Qadd}(u)) < \min(k, |\text{Qadd}(u)|) \) iff the Qsquares \( \text{ATQ}(\text{Qadd}(u)) \) and \( \text{ATQ}(k-\text{Qadd}(v)) \) are disjoint.

Let \( \text{CorrectQadd}(v) \) be the Quadtree address \( v \) would get, when knowing the location of all the nodes in the graph and partitioning the unit square according to quadtree definition in 4.1.

Let \( \text{QADadd}(v) \) be the address \( v \) obtains after QAD algorithm finishes.

Theorem 9 Under the algorithm requirements, after running the algorithm \( \forall v \in V \)
\( \text{QADadd}(v) = \text{CorrectQadd}(v) \).

Proof. Denote \( k = |\text{CorrectQadd}(v)| \), and \( \alpha \) is the leaf Qsquare of \( v \). According to the Quadtree partitioning system definition in 4.1 \( \alpha \) has 1-3 populated k-sibling Qsquares. Under the algorithm requirements all Qsquares are intra-connected, in particular the parent Qsquare of \( \alpha \), meaning \( v \) is connected to at least one of its k-sibling nodes. These two nodes will stop exchanging messages when their addresses will no longer collide; therefore \( (*) |\text{QADadd}(v)| \geq k \) at the end of the algorithm. On the other hand, \( k = |\text{CorrectQadd}(v)| \Rightarrow \) there is no node \( w' \) in \( N \) such that \( \text{lmp}(\text{CorrectQadd}(v), \text{QADadd}(w')) > k \); therefore \( (**) |\text{QADadd}(v)| \leq k \). From \( (*) , (**) \) we can see at the end of QAD algorithm, any node \( v \) obtains its self-address \( \text{QADadd}(v) \) such that \( k = |\text{QADadd}(v)| \) meaning \( \text{QADadd}(v) = \text{CorrectQadd}(v) \).
Chapter 8

Simulations

We compare the performance of the routing algorithms we developed using the C++ based simulation platform OMNET++ 4. For our simulation we used different network models in order to better examine the algorithms by comparing their behavior in the different connectivity characterized networks. We create networks with more than one MultiQuadtree root in order to compare the influence of the number of Qsquare representatives. All simulations start with an existing network, meaning every node knows of its neighbors existence and is able to communicate with them. In addition, each node knows its own coordinates at the unit square. Running the simulation we first randomize the set of $N$ nodes coordination, uniformly distributed in the unit square. Next, we create the network based on the desired number of roots and the network model. Next we choose the traffic pairs; that is 5 percent of $N^2$ uniformly distributed pairs of nodes. This order of the simulation building ensures us that comparing different algorithms will be done on the exact same network topologies, and to the exact same traffic patterns.
8.1 Network Models Used in the Simulations

All the network models are built by first randomizing $N$ nodes, uniformly distributed in the unit square area, constructing the desired number of trees $T$ in order to make sure that an induced MultiQuadtree exists in the network before the algorithms start. Finally, we add the additional edges according to the network model with average degree parameter $D$.

Constructing the trees is made by randomly choosing $T$ roots for the tree to be level 0 representatives, followed by recursively connecting each representative to a randomly chosen representative for each of its populated sublevels. In this way different initial trees can use the same edges in some probability. The final MultiQuadtree discovered by the MQD algorithm described in chapter 6 will include the edges from the initial trees and some of the additional edges we add later according to the desired network model.

Pure tree is the first network model we used, which means the only edges in the network are the initial tree edges. This model is used for comparison and allows us to observe the use of the non-tree edges.

Random Edges is the second network model we used. In this model we add each of the remaining possible edges in the network in a constant probability $p$ such that we achieve the desired approximate average degree in the network $D$.

$p$ is calculated in the following matter:

$$p = \frac{D \times N - 2 \times te}{N \times (N-1) - 2 \times te}$$

where $te$ is the number of initial trees edges.

Explanation: Without existing tree edges, in order to achieve average degree $D$ we would need to have $\frac{N \times D}{2}$ edges in total out of a possible $\frac{N \times (N-1)}{2}$ edges, which makes $p = \frac{\frac{N \times D}{2}}{\frac{N \times (N-1)}{2}} = \frac{D}{N-1}$. But considering the existing $te$ edges, we would like to have an additional $\frac{N \times D - te}{2}$ edges out of a possible $\frac{N \times (N-1) - te}{2}$ edges, which makes $p = \frac{\frac{N \times D - te}{2}}{\frac{N \times (N-1) - te}{2}} = \frac{D \times N - 2 \times te}{N \times (N-1) - 2 \times te}$.
UDG is the third network model we used. In this model we add all of the remaining possible edges in the network that are shorter in physical length than $l$. We determine $l$ such that we achieve the desired approximate average degree in the network $D$.

$l$ is calculated in the following matter:

$$\frac{1}{\sqrt{\pi \times \frac{D}{N}}}$$

Explanation: The intuition for this approximation is dividing the unit square into squared areas with approximately $D$ nodes in each one. There are a total of $N/D$ such squares, each with an area of $D/N$. In order to achieve a circle with this area, the radius has to be $\frac{1}{\sqrt{\pi \times \frac{D}{N}}} = l$.

Distance related is the fourth and last network model we used. In this model we add each remaining possible edge $i$ in the network in a probability $p_i$ that is proportional to the edge length, such that for shorter edges $p_i$ is greater and we wish achieve the desired approximate average degree in the network $D$.

$p_i$ is calculated in the following matter:

$$\frac{\text{Length}(i)^{-2}}{\text{LengthSum}} \times \frac{(D - \frac{te}{N}) \times N}{2}.$$  

where $\text{LengthSum} = \sum(\text{Length}(i)^{-2})$.

Explanation:

$p_i^* = \frac{\text{Length}(i)^{-2}}{\text{LengthSum}}$ is the relative portion of the edge $i$ of all the optional additional edges. $p_i^*$ is greater for shorter edges. In order to achieve average degree $D$, the added degree for each node should be an average $(D - \frac{te}{N})$, bringing the wanted additional edges to $(D - \frac{te}{N}) \times N$. $\sum p_i$ should be $(D - \frac{te}{N}) \times N$ accordingly; therefore $p_i = p_i^* \times \frac{(D - \frac{te}{N}) \times N}{2} = \frac{\text{Length}(i)^{-2}}{\text{LengthSum}} \times \frac{(D - \frac{te}{N}) \times N}{2}$.

8.2 Algorithms Used in the Simulations

All of the routing algorithms we use in this study are variations of the GQG algorithm described in Chapter 5. We created four algorithms characterized by increasing priority to greedy routing rather than tree routing so we could examine and compare the effect of such
change.

**Pure tree** is the first network model we described, which means the only edges that exist in the network are the initial trees edges. This model forces the messages to be routed based on the tree edges only, which lets us compare it to different algorithms that also use other edges. We use this comparison for the same nodes set and locations, and for the same traffic.

**Quadtree (Q)** is the first algorithm we use. It starts message routing in tree mode and never switches to greedy mode. It is important to notice this algorithm does not necessarily deliver the message on the MultiQuadtree edges only, but according to the algorithm 1 delivers the message to the neighbor with the minimal $MQ_{\text{dist}}(w, \bar{a})$ to the destination.

**GQ** is the second algorithm we use. It starts message routing in greedy mode, switches to tree mode at the first local minimum, but never switches back to greedy mode.

**GQG** is the third algorithm we use, which is the exact algorithm described in Chapter 5. This algorithm starts message routing in greedy mode, switches to tree mode at local minimums, and switches back to greedy mode when reaching a node with smaller $G_{\text{dist}}(v, w)$ to the destination than the last local minimum.

**LA GQG** is the fourth and last algorithm we use (Look Ahead GQG). This algorithm starts message routing in greedy mode, switches to tree mode at local minimums, and switches back to greedy mode when reaching a node that has a neighbor with smaller $G_{\text{dist}}(v, w)$ to the destination than the last local minimum. This algorithm gives even greater priority to greedy routing and uses any possible greedy hop.
8.3 Simulation Results

We examined the behavior of the offered routing algorithms described in section 8.2 over the different topology patterns described in section 8.1 focusing on the achieved hop stretch and load balancing.

Figure 8.1: 400 nodes, D=10, Stretch and found path length for the different graphs and routing algorithms.

In Fig. 8.1 the blue bars represent the average path length in hops between two random nodes found by the different algorithms where T=5. The red bars show the addition to the average path lengths found with the same nodes and traffic and same D value, where T=1. The marks on the bars are the average measured stretch corresponding to the graph type.
and algorithm, for both $T=1$ and 5.

In general, Random Edges graphs have the worst stretch, even though the found paths are shorter than the found paths in UDG graphs. This type of graph typically contains many local minimums and very short paths between nodes, but the paths are random and often not greedy. Therefore, both greedy and Q routing do not take the short paths. We also see in this graph the biggest improvement in stretch when creating the infrastructure of $T=5$. This is caused by the addition of tree edges, which create shortcuts for the Q routing algorithm. In the UDG graphs there is a large difference in found path lengths between graphs with $T=1$ and $T=5$. This is caused by the use of the tree edges that can contribute to both Q and greedy routing. However, the shortest paths are also made shorter so the stretch remains similar.

![Figure 8.2: 400 nodes, D=10, T=1 Accumulating Load balance.](image)

Fig. 8.2 presents the cumulative load distribution using different routing algorithms over the Distance related graph. The X-axis is the quantity of messages forwarded by the nodes. This axis is logarithmic, and presents 20 messages bins. The Y-axis shows the cumulative percentage of nodes that forward equal or lower quantities of messages. The
maximal X value is the maximum quantity of messages passing through any node. The standard deviation value is an important value as well; the smaller it is the better the load balancing is. Routing based on the tree edges only is worst with the largest maximum value of 6200, and with very large standard deviation, since most of the about 8,000 messages (5%*400²) traverse through the tree root. Routing in Quadtree algorithm greatly improves the load balancing, with a maximum value of less than 3900 messages, and a 40 percent smaller standard deviation. Using GQG algorithm reduces the maximum value to 810 messages, and the standard deviation reaches 75 percent smaller than in Quadtree routing on the same graph.

Figure 8.3: 400 nodes, D=10, Distance Related graph, Pure Tree/GQG routing load balance ratio.

Fig. 8.3 shows the ratio of load balancing when using the GQG or Q algorithms vs. using routing based on the tree edges only, as a function of the nodes position in the MultiQuadtree network hierarchy. The X-axis is the nodes level in the MultiQuadtree network, and the Y-axis is the ratio between the average quantity of messages a node routes when
using the tree edges only, to the average quantity of messages a node routes when using GQG or Q algorithms. Here we see clearly that the GQG distributes the load better by reducing the load of nodes closer to the roots and adding load to nodes higher in the hierarchy.

Figure 8.4: D=10, Average found path length for different algorithms as a function of the network size.

Fig. 8.4 (C) shows the average hop stretch of the different algorithms as a function of the network size \( N \). Each point on this graph is the average hop stretch of \( 0.05 \times N^2 \) random pairs of source and destination, and for 10 different random networks (node set and topology). Note that the same random networks and traffic patterns were used for testing the different algorithms. We can see the best stretch was achieved by Q algorithm with five roots, \( T = 5 \). The worst was found when routing on the tree edges only with \( T = 1 \). In general, we can see the stretch does not increase significantly when increasing the network size.
Part II

Arithmetic Geographical Coding
Chapter 9

Problem Definition

Geographical routing is a routing method that relies on the knowledge nodes have on their own location as well as their neighbors, and the destination location, in order route messages in the network. This information is coded into the addresses of the nodes; the more information we have about the nodes location the better. On the other hand, it is very important to keep the addresses as short as possible, as they are part of the message header and the routing tables. In other words, an efficient geographical coding system will reduce message overhead, which can increase the network utility as well as reduce the table size held in the nodes, which reduces the amount of memory used.

Our goal is to give a geographical coding system for a set of vertices $V$ of a graph that is represented in a unit square plane $[0,1]^2$. The code for set $V$ is the set of the code words for all $v \in V$ that are unique, bit efficient, and contain the geographical information. We do so in the coding system AGC by giving each vertex $v \in V$ an axis-parallel rectangular area in the unit square plane so that the only vertex it contains is $v$. The rectangular area is described by a two-field binary address, indicating the coordinates of the left bottom corner of the rectangle. The rectangle X and Y sides lengths are represented by the lengths of the corresponding address fields. In this paper we assume $|V| > 1$. 

67
Chapter 10

Addressing Systems

In this chapter we present a Quadtree based geographical coding system called AGC (Arithmetic Geographical Addresses). We present the structure of the codewords, and the relaxations made from the Quadtree coding system in order to reduce the codewords lengths.

10.1 Quadtree Geographical Address System

As mentioned in section 4.1 the idea of Quadtree is to recursively partition a given unit square area into four smaller squares until each square contains only one vertex, i.e., empty sub-squares stop the partitioning.

The $\emptyset$ represents the whole original square area; then each of the four quarters will get the address of the partitioned square with an added suffix to represent which quarter it is. For $v \in V$, let $Qadd(v)$ be the unique Quadtree address of node $v$ resulting from $Quadtree(V)$.

For example, consider the set of nodes in Fig 4.3 (A). For node $d$, its Quadtree address $Qadd(d)$ is ”00”, as it is in the bottom left quarter of the whole square. Node $b$ has the address ”10 01” as it is at the top left quarter of the bottom right quarter of the whole
10.2 Arithmetic Geographical Coding System

We provide an extension to the Quadtree system by allowing two main relaxations. The first is allowing rectangular areas rather than only squared ones. The address length of a node in Quadtree depends on the side length of its square, where smaller squares demand more bits for representation. Quadtree system partitions to axis parallel disjoint squares, which makes the square side length be determined by the minimal side between the X-axis side and the Y-axis one. Therefore, by allowing rectangular areas we allow the enlargement of one of the sides, which saves bits in the address.

The second relaxation is allowing the rectangular areas to overlap as long as there is still only one node in each rectangle. In other words, two or more rectangles can include the same clear of nodes area, which leads to enlargement of the rectangles perimeter, thereby saving bits in the address.

Formal definition and notations are given here: Let \( \{v_x, v_y\} \) be the two-field binary physical address of vertex \( v \). The physical address indicates the binary digits after the decimal point of vertex’s X and Y coordinates in the unit square plane with an unbounded accuracy.

Let \( f \in \{x, y\} \), where \( f \) indicates the field in the address. Let \( v_f^* \) be the arithmetic geographical coding we give to vertex \( v \) at the field \( f \). \( v_f^* = \lfloor v_f \rfloor_s \), where \( s \in \mathbb{Z}_+ \) indicates the number of bits taken from \( v_f \) to create \( v_f^* \).

Let \( |v_f^*| \) denote the size of the address field \( f \), meaning the number of bits used for \( v_f^* \) (equal to \( s \)).

Let \( \{v_x^*, v_y^*\} \) be the two-field code word for vertex \( v \).

Let \( \text{Rect} \left( \{v_x^*, v_y^*\} \right) \) be a function returning the rectangle corresponding to the code word \( \{v_x^*, v_y^*\} \). The left bottom corner of \( \text{Rect} \left( \{v_x^*, v_y^*\} \right) \) is located at coordinates \( \{v_x^*, v_y^*\} \). The rectangle side length in axis \( f \) is \( 2^{-l} \), where \( l = |v_f^*| \). In other words, the length of the binary
address determines the resolution in each dimension; the longer the address field the greater the resolution in the corresponding axis (Fig 10.1).

\[ v_x = 101100... \text{ (Infinitely long)} \]
\[ v_y = 010010... \text{ (Infinitely long)} \]
\[ v_x^* = 1 \]
\[ v_y^* = 010 \]
Rect \([v_x^*, v_y^*]\) is the dark rectangle.
The left bottom corner at \([v_x^*, v_y^*]\) = \{0.5, 0.25\}
The dark rectangle side length in the axis \(f\) is \(2^{-l}\) where \(l = |v_f^*|\).
In x axis \(2^{-1}\) and y axis \(2^{-3}\)

Figure 10.1: Arithmetic Geographical Address example

It is easy to see that if the fields are in of different length we have a rectangle and if the length is equal we have a square.

A vertex’s geographical code word is *symmetric* iff \(|v_x^*| = |v_y^*|\).

**Lemma 10.1** A *symmetric code word can be transformed into a Quadtree address. And a Quadtree address can be transformed into a Symmetric code word.*
Figure 10.2: Symmetric arithmetic address

Proof. Assume $\bar{v}$ be a Quadtree address $p_1, p_2, p_3, \ldots, p_{imax}$ for vertex $v$ such that $p_i$ represent a pair of bits $(p_{ia}, p_{ib})$ in the $i^{th}$ location of the address. Let $QTA(\bar{v})$ be a function transforming a Quadtree address into an Arithmetic geographical coding in the following manner: $QTA(p_1, p_2, p_3, \ldots, p_{imax})$ returns a two-field address such that concatenating all $p_{ia}$ bits generates the $x$ field address, and concatenating all $p_{ib}$ bits generates the $y$ field address (a demonstration is given above).

- $\bar{v}$ describes a square area by definition. $QTA(\bar{v})$ in the described manner will generate a symmetric code word, which we have shown also describes a square area.

- $\bar{v}$ is built of pairs of bits, each pair stands for one partitioning of the current area into four sub-squares, meaning it splits each sides length in two. Therefore, the length of each side of the square area described in $\bar{v}$ is $2^{-imax}$. $QTA(\bar{v})$ generates a two-field binary address, each of them is the length of $imax$; therefore each side of the square area described has a length of $2^{-imax}$ also.

- Finally we compare the left bottom corner of the areas described by both $\bar{v}$ and $QTA(\bar{v})$.

$\bar{v}$ as described above is structured so that:
- If the sub-square is located in the left side half, the first bit of the suffix will be 
  0 (e.g., 00, 01), otherwise the first bit will be 1 (e.g., 10, 11).

- In the same way we can show that for the sub-square located in the lower half of 
  the current square the second bit will be 0 (e.g., 00, 10), and the upper half the 
  second bit will be 1 (e.g., 01, 11).

Therefore we can say the first bit of the suffix determines the starting location of the 
$x$-axis, and the second bit determines the starting location of the $y$-axis, meaning the 
bottom left corner of the square will be at coordinates given by concatenating all $p_ia$; 
therefore it matches the bottom left corner location of the area described by $QTA(\tau)$. □

Lemma 10.2 Symmetric code words for all rectangles in a square plane under the code 
definition create disjoint squares.

Proof. As we have shown, symmetric code words can be transformed into Quadtree 
addresses. Consider two nodes $v, u \in V$ with Quadtree addresses $\overline{v}, \overline{u}$. According to the 
partitioning process of Quadtree, any two Quadtree squares can either be disjoint or one is 
fully contained in the other.

According to the code definition we know the rectangles representing the nodes are dis-
tributed in a way that each rectangle contains only one node. Therefore the squares are 
disjoint. □

10.2.1 Field Separation

We described AGC code words as a two-field pair $\{v_x^*, v_y^*\}$ that well defines a rectangle 
in which $v$ is the only node contained. Practically, in order to use this code word as the 
address of the node $v$, a binary representation of the field separation should be given. In a 
code word such as $\{v_x^*, v_y^*\}$, there are $|v_x^*| + |v_y^*| + 1$ locations in which the field separation 
could potentially be. The actual code word is then the two fields attached together, with
an additional third field that has a fixed size of \( \lceil \log_2(|v_x^*| + |v_y^*| + 1) \rceil \) attached to the end of the code word. This third field contains the actual position of the field separation where 0 is to the left of the first bit. Decoding is simple; on receiving a code word with length \( k \), one has to solve the equation \( x + \lceil \log_2(x) \rceil = k \), which has one solution.

For an example node we look at \( v \) in Figure 10.1 with rectangle \{1,010\}. The number of bits we give for representing the field separation is: \( \lceil \log_2(1 + 3 + 1) \rceil = 3 \) and the actual separation is located at position 1. The code word of node \( v \) is then 1 010 001, which is 1010001.
Chapter 11

AGC Address Distribution Algorithm

In this chapter we describe the AGC address distribution algorithm and prove its correctness. This algorithm receives a set of vertices coordinates in the unit square area and returns the set of code words corresponding to the vertices. Each codeword for a vertex, as explained in 10, represents a rectangle that is contained in the unit square, and contains only that specific vertex. Increasing the length of a rectangle side will reduce the number of bits required for representing it. Therefore, our challenge is to find for each vertex the rectangle that satisfies the conditions above, and takes the smallest number of bits to represent.

11.1 Matching

We define Matching in this section, a definition that will help us later to describe the AGC Coding distribution algorithm. Matching is actually the longest matching prefix two addresses have in a specific field. Formally, let $M_{uv}^f$ denote the longest matching prefix of $v_f$ and $u_f$, $f \in (x,y)$. Matching is a symmetric property $M_{uv}^f = M_{vu}^f$. Let $|M_{uv}^f|$ be $M_{uv}^f$ length in bits. Given vertex $v$, the unit square plane can be partitioned into areas according to the matching values of the vertices contained in them with $v$. 74
Figure 11.1: Two-bit matching to \( v \) partitioning of the unit square. Nodes in the white areas will have values matching \( v \) according to this partition in the corresponding field. Nodes in the dashed areas will have matching values of 3 or more in at least one of the fields.

In Figure 11.1 there is an example of such partitions when partitioning the plane with an accuracy of 2 bits.

Defining \textit{MaximalMatching} and \textit{fielddistance}. Given \( V \), a set of vertices in a complete graph:

- Let \( M_{X}^{\text{max}} = \max \{|M_{uv}^x| u \in N(v)\}, M_{Y}^{\text{max}} = \max \{|M_{uv}^y| u \in N(v)\} \)

- Let \( C_{F}^{\text{max}} = \max \{|M_{v}^{F}\text{max}| v \in V\}, F \in \{X,Y\} \)

- Let \( D_{vu}^f = M_{v}^{F}\text{max} - |M_{uv}^f| \) be the distance between vertices \( u,v \) in the field \( f \in \{x,y\} \).

11.2 The Dplane Data Structure

We build this data structure for each node \( v \in V \) in the algorithm, and use it to find the most bit efficient rectangle for \( v \).
Let Dplane($v$) be a grid data structure for node $v$ in which we locate all $u \in N(v)$ as Dpoints. For each $u \in N(v)$ there is a Dpoint located in the Dplane grid at coordinates \( \{D^x_{uv}, D^y_{uv}\} \). Dplane($v$) is bounded by $M^{Xmax}_v$ and $M^{Ymax}_v$. Dplane is used in the AGC address distribution algorithm. Following is a detailed example of Dplane.

Recall $D^f_{vu} = M^{Fmax}_v - |M^f_{vu}|$.

$v = \{0.758, 0.36\} = \{0.11000, 0.01011\}$ $M^{Xmax}_v = 2$, $M^{Ymax}_v = 3$

$u_1\{0.7, 0.42\} = \{0.10110, 0.01101\}$ \{\|M^x_{vu1}\|, |M^y_{vu1}|\} = \{1,2\} \{\|D^x_{vu1}\|, |D^y_{vu1}|\} = \{1,1\}$

$u_2\{0.53, 0.78\} = \{0.10000, 0.11000\}$ \{\|M^x_{vu2}\|, |M^y_{vu2}|\} = \{1,0\} \{\|D^x_{vu2}\|, |D^y_{vu2}|\} = \{1,3\}$

$u_3\{0.98, 0.07\} = \{0.11111, 0.00001\}$ \{\|M^x_{vu3}\|, |M^y_{vu3}|\} = \{2,1\} \{\|D^x_{vu3}\|, |D^y_{vu3}|\} = \{0,2\}$

$u_4\{0.23, 0.26\} = \{0.00111, 0.01000\}$ \{\|M^x_{vu4}\|, |M^y_{vu4}|\} = \{0,3\} \{\|D^x_{vu4}\|, |D^y_{vu4}|\} = \{2,0\}$

Table 11.1: Creating $Dplane(v)$: Given coordinates of the nodes in $V$ in decimal and binary (left). First we calculate the matching values of $u \in N(v)$ to $v$ (middle row), then determine the maximal matching in each field ($M^{Ymax}_v$, $M^{Xmax}_v$). Then calculate the matching distance (right row). Now we can draw $Dplane(v)$.  

76
We denote $C_v(k, l)$ as a code word for vertex $v$ where $|v_x^*| = k$ and $|v_y^*| = l$. $C_v(k, l)$ is a legal code word iff $Rect(C_v(k, l))$ contains only $v$. (The function $Rect()$ was defined in section 10.2.) One can observe there could be infinite number of legal code words for each vertex, where the different code words have different lengths.

We now show and prove that using Dplane($v$), the AGC algorithm finds for each $v \in V$ the legal code word with the minimal length denoted as $C^*_v(k, l)$, the minimum of all legal $C_v(k, l)$ such that $\min(k + l)$.

Let a legal rectangle in Dplane($v$) denoted as $R$, be a rectangle with the bottom left corner at $\{0,0\}$, and the upper right corner $\{R_x, R_y\}$ such that it is empty of other Dpoints. Formally $\forall u \in N(v)$ $D^x_{uv} \geq R_x$ or $D^y_{uv} \geq R_y$. We define the function $C(R)$ for any legal $R$, $C(R) = C_v(k, l)$, where $k = 1 + M_{\text{max}}^{X} - R_x$, and $l = 1 + M_{\text{max}}^{Y} - R_y$.

A key property for a legal rectangle is given in Lemma 11.1, which holds the reason for the use of the Dplane data structure in the AGC algorithm.

**Lemma 11.1** For any legal rectangle $R$ in Dplane($v$), $C(R)$ is a legal code word.

**Proof.** Let $R$ be a legal rectangle in Dplane($v$) and $C_v(k, l) = C(R)$. Let $D_1$ be a Dpoint corresponding to vertex $u$ in the Dplane ($v$).

![Figure 11.3: Contradicting situation of legal rectangle R in Dplane($v$) (left) and illegal C(R) in the unit square (right)](image)

77
Assume by contradiction \( \text{Rect}(c(R)) \) contains \( u \). Without loss of generality we assume \( x \) to be the axis in which \( D_1 \) is out of \( R \), meaning \((\ast)\)
\[
D^x_{uv} \geq R_x.
\]
From the definition of \( C(R) \), we know \((1)\)
\[
k = 1 + M_{uv}^{X_{max}} - R_x.
\]
And from the definition of \( D^x_{uv} \), we know \((2)\)
\[
D^x_{uv} = M_{uv}^{X_{max}} - |M_{uv}^x|.
\]
Placing \((1)\) and \((2)\) in \((\ast)\)
\[
k \geq 1 + |M_{uv}^x| \iff k > |M_{uv}^x|.
\]
But \( \text{Rect}(c(R)) \) contains \( u \) meaning \( u_x \) is prefixed with \( \lfloor v_x \rfloor \); therefore \( k < |M_{uv}^x| \) which is in contradiction to \( k > |M_{uv}^x| \).

This shows each legal rectangle in \( \text{Dplane}(v) \) can be transformed with the function \( C(R) \) into a legal rectangle in the unit square. Now we shall find the optimal one.

**Lemma 11.2** An optimal code word \( C_v^*(k,l) \) can be generated from a legal rectangle \( R \) in \( \text{Dplane}(v) \) using \( C(R) \) only when one of the following conditions for \( R \) is met:

1. \( R_f = M_v^{F_{max}} + 1 \) and \( R_f = 0 \) where \( f \in \{x,y\} \)

2. Rectangle \( R \) is touching two different \( D \)points, one at the upper side and the other at the right side of \( R \). Formally, \( \exists D_1, D_2 \) in \( \text{Dplane}(v) \) correspond to \( u_1, u_2 \), respectively, such that: \( (u_1 \neq u_2) \land (R_x = D_{vu1}^x > D_{vu2}^x) \land (R_y = D_{vu2}^y > D_{vu1}^y) \)

**Proof.** Let \( R_1, R_2 \) be legal rectangles in \( \text{Dplane}(v) \). Claim: \( C(R_1) \) is shorter code word than \( C(R_2) \) iff \( (R_{1x} + R_{1y}) > (R_{2x} + R_{2y}) \). Let us assume by contradiction a legal rectangle \( R_1 \) such that \( C(R_1) = C_v^*(k,l) \), but \( R_1 \) does not satisfy either of the cases described in lemma 11.2. Therefore, \( R_1 \), in one of the fields - without loss of generality we say the \( x \) field, is bounded by neither the \( M_v^{F_{max}} + 1 \) nor a \( D \)point. We create a rectangle \( R_2 \) with \( R_{2y} = R_{1y} \) and increase \( R_{2x} \) (from the value of \( R_{1x} \)) until we hit the first of the two: \( M_v^{X_{max}} + 1 \) or a \( D \)point. \( R_1 \) is legal, by increasing \( R_{2x} \) in the described manner we did not include any \( D \)point in the area of \( R_2 \); therefore \( R_2 \) is legal as well. According to the claim we can say \( c(R_1) \) has a longer code word than \( c(R_2) \); therefore is not \( C_v^*(k,l) \). ■

78
Algorithm 5 Overlapping rectangles
This algorithm receives a set $V$ of vertices in the unit square area, and returns its Arithmetic Geographical Code, a set of code words $C_v^*(k,l)$ corresponding to all $v \in V$.

Input: a set $V$. Output: CodeSet Arithmetic Geographical Code for set $V$. Let $Dlist(v) = \{D^x_{vu}, D^y_{vu}\} | u \in V \setminus v \}$ a list containing pairs of prefix distances between $u$ and $v$ in both fields for all $u \in V \setminus v$. Let $Possibilities(v)$ be a list containing points indicating the upper right corners of legal rectangles in $Dplane(v)$.

1. For every vertex $v \in V$:
   (a) Create the list $Dlist(v)$.
   (b) Sort $Dlist(v)$ according to $D^x_{vu}$ and for equal $D^x_{vu}$ according to $D^y_{vu}$
   (c) recentX = 0; recentY = $M_{vY}^{max}$+1;
   (d) For every pair $\{D^x_{vu}, D^y_{vu}\} \in Dlist(v)$:
      i. If((recentX==D^x_{vu})&& (recentY≥D^y_{vu}))
         A. recentY=D^y_{vu}
      ii. If ((recentX!=D^x_{vu})&& (recentY ≥ D^y_{vu}))
         A. $\{D^x_{vu}, \text{recentY}\} \rightarrow Possibilities(v)$
         B. recentY = $D^y_{vu}$
         C. recentX = $D^x_{vu}$
   (e) $\{M^{Xmax}_{v}+1, 0\} \rightarrow Possibilities(v)$
   (f) $\{0, M^{Ymax}_{v}+1\} \rightarrow Possibilities(v)$
   (g) Select $\{\alpha, \beta\}$ such that $\max(\alpha + \beta| \alpha, \beta \in Possibilities(v))$
   (h) $C_v(M^{Xmax}_{v} + 1-\alpha, M^{Ymax}_{v} + 1-\beta) \rightarrow CodeSet$
Claim: In this coding system the areas given to vertex \( v \) are independent from areas given to other nodes in \( V \). This attribute allows us to seek a local optimum; when finding the optimal code word for each \( v \) we end up with the optimal for the set \( V \). This algorithm is distributed, based on local knowledge of the neighbors and works in \( O(N(v)\log(N(v))) \) for each vertex \( v \).

### 11.3 Algorithm Correctness

**Theorem 10** This algorithm receives a set \( V \) of vertices in the unit square area, and returns its Arithmetic Geographical Code AGC, a set of code words \( C^*_v(k,l) \) corresponding to all \( v \in V \).

**Proof:** We show that these conditions are met:

1. The list \( \text{Possibilities}(v) \) contains only legal rectangles.
2. The list \( \text{Possibilities}(v) \) contains all the legal rectangles belonging to one of the cases described in Lemma \[11.2\]
3. The list \( \text{Possibilities}(v) \) is not empty.

To prove condition 1 we use induction over the rectangles \( R_i \) in the list \( \text{Possibilities}(v) \) sorted according to \( R_x \) and for equal \( R_x \) according to \( R_y \).

*Base case:* The first rectangle to be examined in the algorithm denoted by \( R_0 \) has its bottom left corner at \( \{0,0\} \), and the upper right corner \( \{0,M^\text{max}+1\} \). We know that all \( D^x_{uv} \geq 0 \); therefore this rectangle is legal.

*Assumption:* Rectangle \( R_i \) is legal and bounded by \( D_a \) from above and by \( D_b \) on the right. See Figure \[11.4\]

*Induction Step:* Proving rectangle \( R_{i+1} \) is legal.
When scanning $Dlist(v)$ sorted according to $D^x_{vu}$, and for equal $D^x_{vu}$ according to $D^y_{vu}$, we know the next Dpoint we check, $D_c$, is such that $D^x_{vc} \geq R_{ix}$. Therefore there are two options:

1. In case $D^x_{vc} = R_{ix}$: (Step (d)i of algorithm 11.2) Scanning $Dlist(v)$ we keep recent$Y$ to be the the minimal value and no rectangle is inserted into $Possibilities(v)$.

2. In case $D^x_{vc} > R_{ix}$ and recent$Y \geq D^y_{vc}$:

   (a) Scanning $Dlist(v)$ in the described manner we make sure there is no Dpoint $D_k$ in $Dlist(v)$ such that $((D^x_{vc} > D^x_{vk} > D^x_{vb}) \land (D^y_{vb} > D^y_{vk}))$, otherwise it would be scanned before $D_c$. In other words, there is no Dpoint inside the rectangle defined by recent$Y$ and 0 in the Y-axis and recent$X$ and $D^x_{vc}$ in the X-axis.

   (b) according to the assumption $R_i$ is legal which makes the rectangle $R_{i+1}$ with the top right coordinates $\{D^x_{vc}, \text{recent}Y\}$ legal $\square$

To prove condition 2, we look at Lemma 11.2. There are two types of rectangles:
• a) The rectangles described by \( R_f = M_v^{F_{max}} \land R_T = 0, \ f \in \{x,y\} \) are inserted “manually” at line 1(e) of 11.2.

• b) The rectangles described by \( \exists D_1, D_2 \text{in } D_{\text{plane}}(v) \) corresponding to \( u_1, u_2 \ (u_1 \neq u_2) \land (R_x = D_{vu1}^x > D_{vu2}^x) \land (R_y = D_{vu2}^y > D_{vu1}^y) \) are as we showed in the proof above, inserted at line 1(d) ii A of 11.2.

Proving condition 3, for every set \( V \setminus v \) with size greater than 1 there is at least one point inserted to \( \text{Possibilities}(v) \) in line 1(e) where \( R_f = M_v^{F_{max}} \land R_T = 0, \ f \in \{x,y\} \).
Chapter 12

Simulation Results

We present three examples in which we compare the two coding systems we discussed in this part of the thesis, Quadtree and AGC coding systems, for different sets of nodes. In each example we show the nodes locations in the unit square, the area represented by the code word of each node, and statistics on the different codes. Notice that according to the coding systems, the Quadtree code word for a node represents a square containing that node only, and the AGC code word for a node represents a rectangular area containing that node only. Above each unit square, we give a histogram showing the distribution of the code word lengths. We also give statistics in the examples about the average and maximal code word length, as well as 95% of the maximal length, which means we ignore the maximal length 5% of the code words in order to ignore the tail of the histogram, and take into account the main mass of code words.

Section 12.4 shows an extended simulation that shows the average and maximal code word length in both Quadtree and AGC coding systems for random sets of nodes of different sizes.
12.1 32 Nodes Example

Figure 12.1 is a demonstration of a 32-node set given codes by both algorithms; the nodes coordinates were uniformly distributed over the unit square plain. The example demonstrates the difference in the areas given to the nodes in the different coding systems, which are then transformed into the code set. On the left side we can see the square partitioning of the Quadtree coding system, and on the right is the rectangular areas of AGC coding system. We can see the improvement in maximal code word length and in the 95% measurement.

<table>
<thead>
<tr>
<th></th>
<th>Quadtree</th>
<th>AGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average code word length</td>
<td>7.75</td>
<td>7.875</td>
</tr>
<tr>
<td>Maximal code word length</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>95% maximal code word length</td>
<td>12</td>
<td>9</td>
</tr>
</tbody>
</table>

Figure 12.1: 32 uniformly located nodes coded in Quadtree and AGC coding systems. The areas in the unit square are represented by the nodes code words and a histogram of the code
12.2 512 Nodes with Padding Example

As explained, the geographical addresses are ultimately used in message headers and in the routing tables the nodes hold, in order to route the messages to their destination. The address fields have a fixed size that has to be sufficiently large in order to hold the maximal address length; therefore the address size allocated in the routing table and messages are according to the maximal code word.

In this example we show the areas represented by the *padded* code words. Meaning in each coding system, after calculating the code words for all nodes, we use the maximal code word length as the length of all the code words in the set. This way we take advantage of the full size allocated to the address, which allows us to hold more information about the location of the node.

Figure 12.2 is a demonstration of a 512-node set given codes by both algorithms. The nodes coordinates were normally distributed over the unit square plain. The example demonstrates the difference in the areas given to the nodes in the different coding systems, which are then transformed into the code set.

We can see the significant improvement in maximal code word length.

<table>
<thead>
<tr>
<th></th>
<th>Quadtree</th>
<th>AGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average code word length</td>
<td>11.625</td>
<td>12.6895</td>
</tr>
<tr>
<td>Maximal code word length</td>
<td>26</td>
<td>18</td>
</tr>
<tr>
<td>95% maximal code word length</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>
Figure 12.2: 512 normally located nodes coded in Quadtree and AGC coding systems, the areas in the unit square represented by the nodes code words, and a histogram of the code.

12.3 World Weather Stations Example

This is a demonstration of coordinates of 17,168 world wide weather stations taken from the Mathematica-7 data base and normalized into a unit square area.

<table>
<thead>
<tr>
<th></th>
<th>Quadtree</th>
<th>AGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average code word length</td>
<td>21.016</td>
<td>15.0455</td>
</tr>
<tr>
<td>Maximal code word length</td>
<td>38</td>
<td>31</td>
</tr>
<tr>
<td>95% maximal code word length</td>
<td>28</td>
<td>18</td>
</tr>
</tbody>
</table>
Figure 12.3: 17,168 world wide weather stations nodes coded in Quadtree and AGC coding systems, the areas in the unit square represented by the nodes code words, and a histogram of the code
12.4 Maximal and Average Code Word Length in Random Networks

The following graphs show a comparison of four coding systems. The first gives the nodes serial numbers, which demand the minimal length of $\log_2(N)$ bits for each code word. This efficient coding is presented for comparison purposes as it does not consist of any location information and cannot be used in geographical routing. The two other systems are Quadtree as discussed earlier, and our AGC system including the comma information. Each point in the following graphs represents an average of 10 different two dimensionally normally distributed sets with an average at the center (0.5) and standard deviation of 0.1.

Figure 12.4: Average code word length for a growing set of nodes

In Figure 12.4 we see the average code word length in the Quadtree and AGC systems are very similar.
In Figure 12.5 we see the maximal code word length in the Quadtree and AGC systems. This is a far more significant comparison, as the maximal code word length will eventually determine the address field size in both the routing tables and in the messages header. We can see a growing difference between the systems as the set size grows.

Figure 12.6: Improvement in maximal code word length when using AGC
In Figure 12.6 we see the growing percentage of improvement in the maximal code word length with the growth of the set size. We ran the simulation for sets in powers of 2 sizes, and we can see that for $2^{14}$ we gained 40% reduction in the maximal code word size. This result is very important to us as it shows we can deploy this system in very large networks, and derive greater improvement.
Chapter 13

Conclusions and Future Work

In the first part of this thesis we offer a Quadtree based network hierarchy and a geographical address system that are then used in two routing algorithms with guaranteed delivery. In the results reported here our local, distributed algorithm presents low routing stretch and good load balancing. Future research is in place to determine the use of Quadtree based networks in the context of data centers, and comparison to the existing data centers routing methods, as well as in the context of wireless networks and peer-to-peer networks. We examine the possibility for the MultiQuadtree infrastructure to contain virtual edges, meaning edges that are made of a set of physical edges. This way the infrastructure condition is more flexible and only the connectivity has to be guaranteed. Two more topics for further research are disappearing and reappearing nodes, mobility, and applying this method for a 3D volume.

In the second part we offer an arithmetic geographical coding system that is based on relaxation of the well known data structure called Quadtree. We give here simulation results that show growing improvement in the length of the maximal code word in our coding in comparison to Quadtree coding, as a function of the network size. Future research in this field is to define the patterns in which the differences from Quadtree coding are the greatest and the smallest, and to research applications for the offered method.
Bibliography


92


