Université d'Ottawa · University of Ottawa

A Uniform Randomized Routing Algorithm

by

Felipe Contreras

A thesis
presented to the University of Ottawa
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy, Computer Science

School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada

The Ph.D. program in Computer Science is a joint program with
Carleton University. administered by the Ottawa-Carleton
Institute for Computer Science

Abstract

Given a set of routes between pairs of sites over a communication network, the traffic load of a link measures the number of routes using it. We analyze traffic load for some randomized local routing algorithms, some of which assume geometric information on the network. We also propose a uniform randomized routing algorithm generating uniform distributed routes between a pair of sites where only source, destination and current neighbor information are available.

Thanks

Contributions in this work could not have been possible without Jorge Urrutia (UNAM). I am indebted to his insights and thorough supervision. The author wishes to thank Carlos Zamora (UNAM-Mexico) for his many suggestions on the results presented in this work. I would like to thank also to Ivan Stojmenović (University of Ottawa) for his comments on this work and initial suggestions, and to Margaret Urrutia for the style and english revision of the thesis. The author received support from CONACyT-México Scholarship No. 110147.

Contents

List of Figures

Chapter 1

Introduction

The development of new technologies such as Global Positioning Systems (*GPS*), and continuous decreases in the price of hardware for sensors, transmitters, processors and even transportation devices, has motivated the development of routing techniques which take new factors into consideration, such as the number of sites or their geographic location in the communication network. It has also stimulated a revival of problems previously solved in other fields such as mathematics (graph theory, geometry, topology, etc.).

Motivation to develop new routing schemes comes from a variety of sources. For example, applications such as weather forecasting or fluid movement modelling, where massive computations are required to achieve faster and more accurate results, can now use clusters of personal computers instead of expensive supercomputers. One of the problems, however, with computer clusters is that communication between nodes has to be as efficient as possible because it is the slowest part of the process. Nevertheless, routing methods such as those currently used for the Internet leave ample scope for

new algorithms.

At first glance the problem may seem simple; two sites wish to communicate using intermediate sites, or an item needs to be transported from one place to another through a road network. However in large scale networks there is a large number of variables influencing the choice of path; traffic congestion, distance traveled, and many more, that make the problem very difficult to solve. Some of the theoretically hard problems have been only partially solved or solved in greedy ways in order to have a solution for the emergent technology just before (or sometimes even after) it hits the market.

It is evident that communication and transportation are necessary to everyday life, for whenever we have at least two different options for moving information or items from one point to another, we have to establish a non trivial routing scheme.

Definition 1.1 *The communication network can be seen as a set of entities capable of communicating information or translating objects together with their communication links.*

Sites and links may be static or incessantly moving but fixed for a period of time long enough for a routing scheme to trace a path and establish communication.

We can define a *Routing Scheme* as a method to describe the path in a network that a particular message or item will follow between each pair of source-destination entities or sites.

Communication networks, both permanent and temporary, are found everywhere, from city streets to cellular telephones; from interconnected processors in a parallel computer to the Internet.

The simplest routing is when the two sites willing to communicate are linked by only one connection. This is the case for most small to medium sized networks in which there is no route redundancy. For example, the operator in a small hotel may be able to connect telephones of two guests using a manual switchboard: that is, any pair of guests can communicate directly—a simple network.

Harder problems arise in small town telephone networks, where connections may be still handled by an operator-controlled central. For larger cities however, different solutions are required, calling for a hierarchic structure. In this type of structure, the phone lines of a small area are grouped into a local switchboard, local switchboards in turn are grouped into area switchboards and so on. Communication between two users is usually established by a bottom-up approach that moves upwards until communication is established (see Figure 1.1).

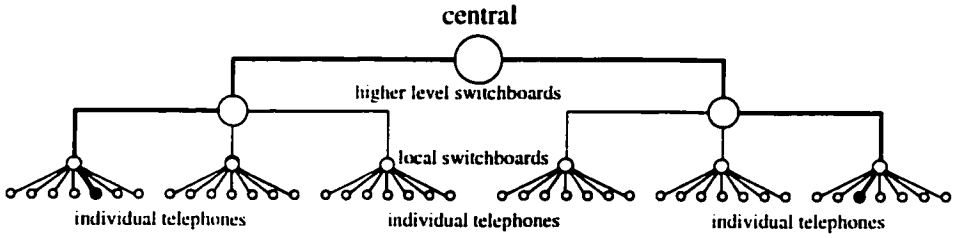


Figure 1.1: Communication through a tree. Only one path is possible between each pair of sites

If sites are linked without forming cycles, as is the case for trees, then only one path is possible between two nodes.

However the problem becomes more complicated for larger telephone networks. For example, can the number of simultaneous calls passing through

the links between two levels of switchboards be estimated?

In more complex networks, it is not clear what parameters should be optimized to route messages. For example, if shortest paths are always used to establish a communication between all pairs of nodes, serious congestion problems may arise. From our experience of driving in large cities, we know that sometimes the best way to reach a destination is by choosing a path longer than the shortest path, in which the traffic is not congested. Communication networks are no exception to this kind of problem, and a good routing scheme should therefore take factors such as shortest paths, traffic congestion at the nodes, the edges of a network and edge and node failures into consideration. Designing such routing schemes, even for specific networks, is a monumental task which defies some of the most creative minds of our time.

Moreover, the constant evolution of computing and communication technology demands the design and analysis of new routing schemes devised to best take advantage of their characteristics.

Many routing schemes have been developed in a natural way. Whenever there is more than one routing option from a to b, *exhaustion* methods can always be applied. Round-robin, described here as an example, is one such method. It consists of trying all possible routes from a to b, one per message, following a fixed order, then repeating the routes until all messages are sent.

Round-robin is used as part of many low-level routing schemes. It has the advantage of distributing traffic uniformly among the edges incident to the origin node and the disadvantage of having to know or calculate all possible routes each time. Another disadvantage is its poor ability to deal

with failures and loops. A flawed link will not be eliminated: it will always be reconsidered. Round-robin can be useful, however, when applied on top of other routing schemes. The first scheme determines the routes and Round-robin decides which one to take next; for example, it can be applied to de-randomize routing schemes. We will not discuss Round-robin further since it is a very simple scheme. It is mentioned at this point however, due to its widespread use. IPv4 Internet routers [1], for example, use a Round-robin method at a certain stage of routing to decide which router can be used for the next hop in the approach to the destination.

1.1 Background

Methods to obtain paths or shortest paths between sites of a network, such as Bellman-Ford and Dijkstra [6], can be found in the literature. These algorithms have serious drawbacks from a practical point of view. They assume the existence of an entity with knowledge about the whole network in order to calculate the routes. They also assume that all networks are stable and that no edge failures occur. In addition, factors such as rapid growth of communication networks or unreliable sites, as well as the ability of networks to interconnect, pose major problems to the implementation of many of the theoretically feasible and possibly optimum schemes. Today, variations of these methods, known as "vector distance" methods, are widely used in a variety of applications.

In this thesis, however, we are interested in distributed schemes (also called *localized* (see for example [26]), where there is no single central node

to store all information about the network. In fact, for practical applications such as the Internet, it is infeasible to keep detailed information about the whole network for the sole purpose of routing a message. Instead, each node may keep a certain amount of information about the network, and the message traversing the nodes may carry more pieces of information to help determine its path. Centralized algorithms such as Dijkstra's are good for small networks, but as size and complexity increase, the problem needs to be looked at from a different point of view.

This work will deal mostly with local routing schemes.

Definition 1.2 *A routing scheme is called local (or localized) if, in order to find the route from source to destination, the only factors which the scheme takes into consideration are the following:*

- *A message contains a constant amount of space to store information such as, for example, the address of its starting and destination points, and a constant additional amount of information, to be used to navigate to the destination.*
- *Local information stored at each vertex of the network, containing information on its neighbors.*
- *The message cannot change the information stored at the vertices of the network, nor can it leave traces at a visited node.*

The last restriction is necessary in large communication networks; the amount of information passing through them is very high, and it is not feasible to keep track of which information has visited the node.

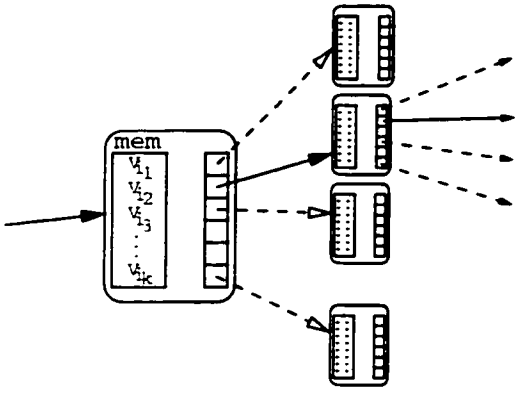


Figure 1.2: Intermediate vertex in a route using some local routing scheme; "mem" and addresses of neighbors are the only information available to it

---

A local routing scheme (see Figure 1.2) does not assume knowledge of global factors such as graph topology or connectivity. Nodes are not allowed to store large amounts of information nor in fact any node information other than the addresses of the neighboring nodes. This restriction is imposed to reduce the complexity of sites, thus reducing costs of practical implementations of the scheme.

An important property of the schemes treated in this thesis is sometimes overlooked by current methods; if the scheme defines at least one path for every pair of nodes of our network, then it is said to *guarantee delivery*.

Finally we say that a scheme is *randomized* if at least part of the decision of which neighbor a node next sends a message to depends on a random, usually Bernoulli-distributed, variable (also called *oracle*). Otherwise, the scheme is called *deterministic*. We will formalize these terms in Chapter 3.

1.2 Existing solutions and motivation

Although it is true that Internet messages, whether traveling via a physical wire or wireless link, travel at the speed of light, it would be misleading to consider this as the speed of the network. Technological factors such as satellite links, where distance is a factor, or even more significant, the number of routers and signal enhancers between nodes and the speed with which they pass messages to their neighbors, would give a better indication of the network speed.

In fact, the speed with which a specific model of router handles messages depends on several factors, and is not constant over time. Among these factors are the methods used to handle messages. For example, it is well known that the IPv4 (and IPv6) Internet protocol currently in use searches over routing tables. A Bellman-Ford-like algorithm is applied to construct these tables, which show how to get to the *next hop* in the network in order to reach the message destination (see [1, 11]). Such tables indicate known routes "nearer" to the destination, based on IP address and on fixed known destinations.

Examples of distributed networks include *ad hoc* or *spontaneous* networks.

Definition 1.3 *Ad hoc or spontaneous networks are those in which the sets of sites and links change over time as nodes appear and disappear freely from the network when a user site connects or disconnects (see for example [5, 21]).*

Networks of cellular phones involve two types of nodes, fixed antenna towers communicating with each other, and personal phones, whose position and activation cannot be predetermined. Low Earth Orbit satellites define

systems involving dozens of satellites. With a limited time to contact neighbors in different orbits, the edges of a network spanning the Earth are made up of only of satellites located close enough together.

1.3 Conditions, assumptions and limitations

If we represent the entities as *sites* or *nodes* of a graph, then two nodes have a *link* or *edge* joining them if the corresponding entities can communicate with each other directly with no intermediate entities intervening. Two sites can thus communicate either directly (*adjacent* nodes) or through a series of *hops* over nodes between them. We assume that the unit of communication between nodes is the *message*, which in practice can be a transportation device such as a car, a communication package, or a piece of information. Thus communication between two nodes must be established by a *path* or *route* joining adjacent nodes from source to destination.

Definition 1.4 *A path is called shortest if it uses the minimum number of hops to go from source to destination. There may be more than one shortest path.*

In this thesis we will ignore some factors which are relevant for real life applications. These restrictions will allow us to better understand the problems studied here. We will assume that the following restrictions apply to our networks: the length of the message (assumed 1), the time a node needs to send, receive or redirect a message (assumed 0), the number of messages that a node can send or redirect at the same time (∞), the number of messages that an edge can carry (∞), the time a message takes to traverse a link (1),

probability of message loss or incompletion (0), probability of link or node failures (0), existing *traffic* or *load* (number of messages) on the link (0). We will discuss the practicality of some of these assumptions at the begining of next chapter.

Whenever possible we will present localized schemes, though due to network complexity, some of our schemes will not comply with the definition. We will mention that aspect in our discussion of the scheme. However our schemes will always describe shortest paths, and delivery is guaranteed.

Definition 1.5 *For our purposes, we assume that during a period of time, every pair of nodes exchange a message. We then measure the traffic that uses each edge in the network, and define this as the traffic load of the edges.*

One aspect studied here is that of designing routing schemes that distribute traffic load as evenly as possible among the edges of a network. In some cases (random paths), our algorithms may solve the problem of bypassing nodes or edges of a network where a failure may occur.

1.4 Contributions

There are several problems that can be studied when routing on a communication network. For example, we would like to lower the traffic over links when all pairs of nodes send messages between them. Is it possible to distribute the traffic perfectly among the edges of a particular network, or at least, can examples of networks where this is possible be shown? How difficult is it to distribute the traffic as uniformly as possible and what is a good

metric for this distribution? Is it better to try to route on fixed predefined paths or to introduce random but meditated decisions? What would be the best routing to apply so that links are used optimally? While the complexity of these questions still remains open for most communication networks, in this work we approach them from two points of view; *global traffic reduction* and *uniform route selection*.

By global traffic reduction we mean taking advantage of the "shape" of the network to mathematically design the routing scheme so that every link in it receives nearly the same amount of traffic. We show that this is not always possible, but also present some general examples where it is.

Another approach, uniform route selection, proposes distributing the traffic uniformly between pairs of nodes.

We show both deterministic and randomized schemes and explain the advantages and disadvantages of each, choosing those which are more robust in the sense of guaranteeing message delivery even through link or node failures.

This thesis thus proposes solutions to alleviate the problem of traffic distribution by two approaches; almost perfect routings (Chapter 4) and uniform route selection (Chapter 5).

Another main result of this thesis (detailed in Chapter 6) proves that the load distribution of randomized routing schemes is *stable*. In simpler terms, this result means that not many simulations are required to measure traffic over each edge, since it quickly approaches expected traffic, which in turn, can be calculated by other means.

As a collateral result, in Chapter 5 we also show an interesting general-

ization. I. Stojmenovic posed the question of finding local routing schemes for networks embedded in R^d, $d \geq 3$. Although examples of such schemes in the literature were not found, we generalize our results for integer lattices to the d-dimensional case.

In Appendix A we show graphical examples of the traffic load for various schemes using the mesh as the underlying graph. We describe some pertinent aspects of their shape. These graphics provided us with motivation for the result in Chapter 6.

Finally we want to mention that this area of research cannot be considered exhausted, as the present work is only part of a continuous project on routing which applies non-standard techniques such as Computational Geometry.

Chapter 2

Literature review

Routing in communication networks has been, and continues to be a major research topic in Computer Science. A number of recent papers deal with implementation and simulation issues for different types of communication networks. For example, in cellular telephone networks a major problem arises from the ability of users to move freely within a certain region [13, 19, 20]. In other papers the authors study problems arising from the capacity and type of communication links, e.g. radio transmission used to communicate with satellites, or the use of optical or copper wires. In the case of radio communications, one has to be careful about the choice of the frequencies used to transmit simultaneously, making sure that transmitters too close together do not use the same frequency, creating interference.

As mentioned in the previous chapter, we impose some restrictions on our schemes in order to be able to deal with the problem and to concentrate on the communication aspects of the routing. However it is important to mention that some of these restrictions represent no obstacle for real life

implementation of the schemes presented in this work, nor perhaps in other kinds of networks not dealt with here; the latter may be also a subject for further research in the future.

For example, instead of instantaneous message delivery, a constant value could be calculated as the weight in time that a certain application takes to deliver a message under certain conditions of the network. We can use that constant value as a factor in the time complexity of the algorithm applied. Of course, in the asymptotic analysis, the assumption of constant values amounts to no change. However this amount of time may imply the need of new devices to handle the messages, such as queues at each site, and queues in turn may imply other complexities, so that our routing schemes become only a layer in the overall scheme.

In the remainder of this chapter we review schemes mainly directed at the routing part of the problem, showing some of the ideas we have found most useful for this task, giving the reader a more complete perspective of the flavour of the results presented in the remainder of this work.

In general, greedy routing algorithms try to reach their destination based on local decisions and on information available at the nodes of the network, choosing at every stage the move that appears best. We present a greedy algorithm to route on lattice networks, i.e. networks whose vertices have integer coordinates, two of which are connected if their distance is one. Typically when moving from a point with coordinates (i, j) to a point with coordinates (k, l) we first move horizontally until we reach the point (k, j), and then vertically until we reach (k, l).

A typical routing algorithm on hypercubes can be described briefly as

follows. Suppose we want to move from a point with coordinates $(i_1, \ldots, i_k)$ to $(j_1, \ldots, j_k)$. Find the first index m such that $i_m \neq j_m$. First move to $(i_1, \ldots, i_{m-1}, j_m, i_{m+1}, \ldots, i_k)$. Proceed recursively from this point until we reach $(j_1, \ldots, j_k)$. As we will see later, this algorithm, aside from its simplicity, has some additional properties that makes it particularly useful in networks.

Interval routing assigns, whenever possible, labels to the nodes of the network so that at each site, an interval is assigned to each edge incident at this node which determines the next node to visit in order to reach the given destination.

Finally, compass routing and its variations originate from the intuitive idea of following the edge whose direction points most nearly in the direction of the destination.

2.1 Greedy Routing

Definition 2.1 *A mesh or integer lattice of size $n \times m$, denoted $\mathcal{L}_{n\times m}$, is the geometric graph*[1] *$G(V, E)$ having $V = \{(i, j) \mid i \in \{0, 1, \ldots, n-1\}, j \in \{0, 1, \ldots, m-1\}\}$. where the neighbors of each vertex differ by 1 in any of its coordinates. $\mathcal{L}_{n\times n}$ can also be denoted by $\mathcal{L}_n$.*

Possibly the simplest and most used local routing scheme on the mesh is the following *Greedy* Algorithm.

Algorithm 1. Greedy(a,b)

[1] A *Geometric Graph* is an embedding of a graph in the plane whose nodes are points and whose arcs are straight edges.

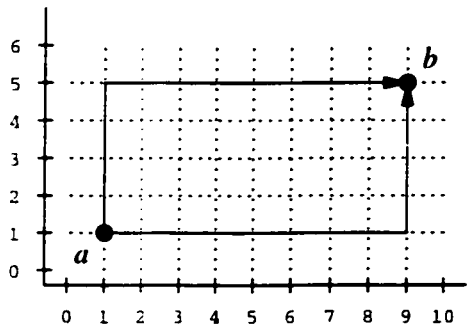


Figure 2.1: Possible routes generated by Randomized Greedy Routing Algorithm from a to b

---

1. Let $a = (a_x, a_y)$, $b = (b_x, b_y)$, $\pi \leftarrow \{a\}$
2. $\sigma_x \leftarrow \text{sign}(b_x - a_x)$; $\sigma_y \leftarrow \text{sign}(b_y - a_y)$
3. While $|b_x - a_x| > 0$
   (a) $a_x \leftarrow a_x + \sigma_x$
   (b) Add the new a to π
4. While $|b_y - a_y| > 0$
   (a) $a_y \leftarrow a_y + \sigma_y$
   (b) Add the new a to π

•

Here sign(x) returns -1, 0 or 1 according to the sign of x.

In other words, to find a route from site a to site b, the Greedy Algorithm first traverses nodes horizontally from a to the column of b, then traverses nodes vertically to reach b.

Alternatively the Greedy Algorithm could have started traversing nodes vertically from a to the row of b, then horizontally to reach b (exchange steps 3 and 4). If we decide randomly whether to start horizontally or vertically from a we have the *Randomized Greedy* Routing Algorithm.

In Figure 2.1. if $a = (1, 1)$ and $b = (9, 5)$ then the possible paths generated by the Randomized Greedy Routing Algorithm over $\mathcal{L}_{11\times 7}$ are

$$\pi_h = \{(1,1),(2,1),\ldots,(9,1),(9,2),\ldots,(9,5)\}$$

and

$$\pi_v = \{(1,1),(1,2),\ldots,(1,5),(2,5),\ldots,(9,5)\}$$

As mentioned above, the Greedy Algorithm has been studied previously in papers such as [14]. In this paper, the authors use a model with queues at each site in which messages are stored if the needed link is busy. Then messages traveling farthest in the link direction are popped off of the queues first and forwarded. They give an upper bound on the delay of a packet to reach its destination and on queue length in terms of the number of packets and size of the lattice. They also explore other variations of the Greedy Algorithm using queues, over cycle and torus graphs.

2.2 Dimension walking

Definition 2.2 *A hypercube $\mathcal{Q}_d$ in d dimensions is the graph with vertices $a_0, a_1, \ldots, a_{2^d-1}$ and edges $a_i a_j$ such that $|i-j| = 2^t$, for some $t \in \{0, 1, \ldots, d-1\}$ and d a positive integer.*

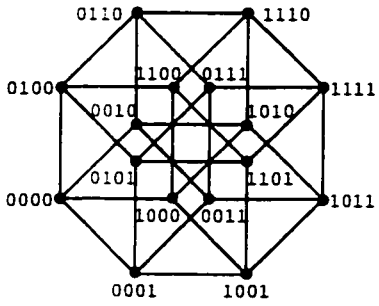


Figure 2.2: Hypercube $\mathcal{Q}_4$. Numbers are binary representations of vertex indices

---

Definition 2.3 *The binary representation $b_{d-1}b_{d-2}\cdots b_0$ of index j of a vertex has binary digits (bits) $b_i \in \{0,1\}$ such that $j = \sum_{i=0}^{d-1} 2^i b_i$.*

Since indices of consecutive vertices in a path have to differ by one bit, we can define a routing scheme by defining the order in which bits are turned *on* or *off* on consecutive vertices. This provides $d!$ (d factorial) different versions of the scheme over $\mathcal{Q}_d$.

For example, let $\mathcal{A}$ be the scheme turning *on* or *off* bits from right to left whenever necessary to reach the destination.

Algorithm 2. Dimension walking (a,b)

1. Let $a_{d-1}a_{d-2}\cdots a_0$ and $b_{d-1}b_{d-2}\cdots b_0$ be the binary representations of sites a and b respectively.
   Let $i \leftarrow 0$ and let $\pi \leftarrow \{a\}$.
2. While $i < d$ and $a_i = b_i$
   $i \leftarrow i + 1$

3 If $i < d$ then $a_i \leftarrow b_i$

Add $a = a_{d-1}a_{d-2}\cdots a_0$ to π

4. Go to Step 2 while $i < d - 1$

•

Algorithm $\mathcal{A}$ will be called *dimension walking* over $\mathcal{Q}_d$.

For example in Figure 2.2. dimension walking from a_{1101_b} to a_{0110_b} will define the path $\pi = a_{1101_b}a_{1100_b}a_{1110_b}a_{0110_b}$.

If i and j binary representations differ in k bits, then the distance from a_i to a_j is k. It is easy to see that dimension walking will always produce shortest paths.

We will see an important property of dimension walking in Section 3.1.

2.3 Interval Routing

In [22] the authors propose a class of routing algorithms called *Interval Routing*. This was extended in [27], and [9] surveys the topic.

Whenever possible, Interval Routing uses a *labeling* $v_1, v_2, \ldots, v_{n-1}$ for the vertices of the underlying graph $G(V, E)$, such that for each directed edge $e_{i,j} = v_i v_j$, there is an interval $[a_{i,j}, b_{i,j}] \subseteq \{0, 1, \ldots, n-1\}$ with the following properties:

1. $[a_{i,j}, b_{i,j}] = \{a_{i,j}, a_{i,j} + 1, \ldots, a_{i,j} + t = b_{i,j}\}$ where addition is taken modulo n.

2. $[a_{i,j}, b_{i,j}] \cap [a_{i,k}, b_{i,k}] = \emptyset$. for all edges $e_{i,j} \neq e_{i,k}$ incident to v_i

3. $\bigcup_{\substack{j \\ v_i v_j \in E}} [a_{i,j}, b_{i,j}] = \{0, 1, \ldots, n-1\}$

4. If $k \in [a_{i,j}, b_{i,j}]$ then there exists a shortest path from v_i to v_k using edge $e_{i,j}$

It has been shown [22] that trees support Interval Routing. We now present a method to label the spanning tree $\mathcal{T}$ of a graph:

Algorithm 3. Labeling scheme for Interval Routing of the spanning tree $\mathcal{T}$ of a graph

1. Label vertices with $v_0, v_1, \ldots, v_{n-1}$ in postorder traversal fashion, that is, descendants of a vertex receive lower index labels than the vertex.

2. Directed edge $e_{i,j} = v_i v_j$ receives interval $[a_{i,j}, b_{i,j}]$, where $i < j$ and

   $a_{i,j} = \min\{k \mid v_k$ is in the subtree of $\mathcal{T}$ rooted at v_j excluding edge $e_{i,j}\}$

   $b_{i,j} = \max\{k \mid v_k$ is in the subtree of $\mathcal{T}$ rooted at v_j excluding edge $e_{i,j}\}$

3. Edge $e_{j,i}$ receives interval $[b_{i,j}, a_{i,j} - 1]$

4. Root index may be added to the interval of one of its outbound edges as a final step.

•

Interval Routing speeds up the process of finding the next vertex to visit from vertex v_i because it uses a binary search on the sorted intervals.

Preprocess 1. (Finds and sorts intervals at each vertex)

For each vertex v_i in G:

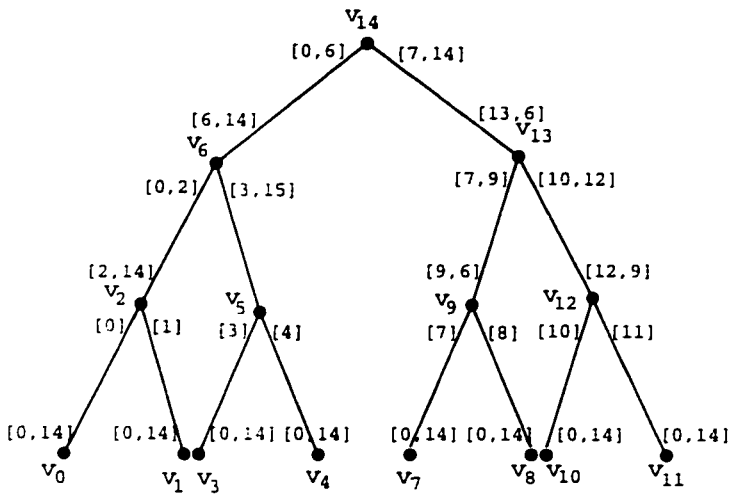


Figure 2.3: Spanning tree $\mathcal{T}$ of a graph and its labeling for Interval Routing

---

1. Let $I_i = \{[a_{i,t}, b_{i,t}] \mid v_i v_t \in E\}$, the set of intervals for edges incident to v_i.

2. Sort I_i.

•

Algorithm 4. Interval Routing from a to b

Let $a = v_0$, $b = v_k$, $i \leftarrow 0$, $\pi \leftarrow \{a\}$

1. Find j such that $k \in [a_{i,j}, b_{i,j}]$ (binary search of k on I_i)

2. Add v_j to π

3. $i \leftarrow j$

Repeat until $v_j = b$

•

Interval Routing eliminates the space required for the storage of *routing tables* (see [8, 10]). the method formerly used, and still used on networks such as the Internet.

Our routing algorithm (Algorithm 10) uses the same ideas as Interval Routing, since it does a binary search of the information stored at incident edges of the current vertex to derive the next neighbor in the route. Another algorithm using these ideas is Compass Routing.

2.4 Compass Routing

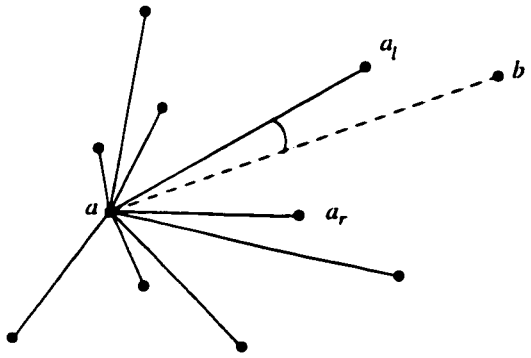


Figure 2.4: Finding the best next vertex from a with Compass Routing

Compass Routing, first described in [16, 24], is another example of a local routing algorithm on a geometric graph. This algorithm uses the additional information of site coordinates to calculate angles which are used to choose the next site. Angles and arcs are expressed counterclockwise (positive direction), unless specified otherwise.

Algorithm 5. Compass Routing from a to b

1. $\pi \leftarrow \{a\}$

2. While $a \neq b$

   (a) Let $\{a_1, a_2, \ldots, a_k\}$ be the neighbors of a

   (b) Find indices l and r such that a_l determines the minimum positive angle $\measuredangle baa_l$ and a_r determines the minimum positive angle $\measuredangle a_r ab$. Let a be a_l or a_r, whichever determines the smaller angle. Ties are broken randomly.

   (c) Add the new a to π.

•

Vertices a_l and a_r are the first to the left and the first to the right of directed segment ab respectively (see Figure 2.4). We choose as next vertex the one making the smallest angle with segment ab.

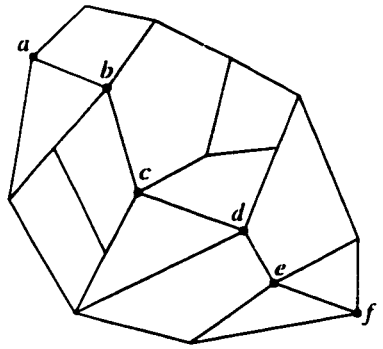


Figure 2.5: Routing from a to f

For example, to find a route from site a to site f in Figure 2.5, we traverse nodes $abcdef$.

In a sense the motivation for Compass Routing Algorithm comes from Interval Routing since it performs a local search among the candidates for the next node to visit, based on the angle from the current node to the destination.

Unfortunately Compass Routing does not guarantee delivery for several important classes of underlying graphs. We can find simple counterexamples in graphs with low connectivity, but even worse, in graphs where all the faces are convex as in Figure 2.6. This graph is constructed from two regular hexagons, the inner one slightly tilted. When routing from a to b, Compass Routing would produce the loop

$$v_1 w_1 v_2 w_2 v_3 w_3 v_4 w_4 v_5 w_5 v_6 w_6 v_1$$

Instead of hexagons, any other regular polygon with more than 3 sides can be used.

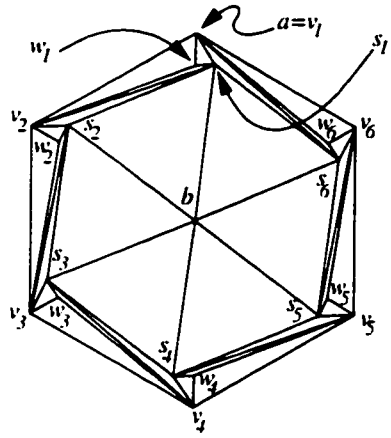


Figure 2.6: Compass Routing alone does not guarantee delivery

In this figure, line segment $v_i b$ contains w_i, and is perpendicular to segment $s_i w_i$. Therefore with a as current vertex, angle $\angle w_1 a b$ is smaller than

$\measuredangle bas_1$; with w_1 as the current vertex, angle $\measuredangle v_2 w_1 b$ is smaller than $\measuredangle bw_1 s_1$; we reach v_2 and the process repeats.

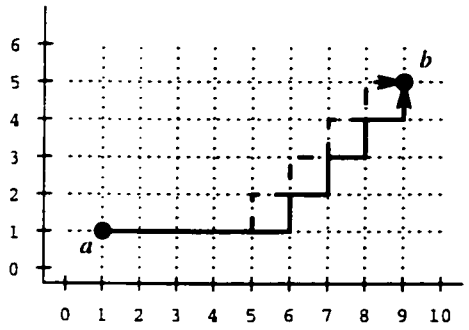


Figure 2.7: Compass Routing paths over $\mathcal{L}_{11\times 7}$

It is easy to see that Compass Routing works on integer lattices (see Figure 2.7). There is, however, an important family of graphs where Compass Routing guarantees delivery, called Delaunay triangulations.

Definition 2.4 *A triangulation of a finite set of points is a subdivision of the plane by the maximum number of non-overlapping line segments joining pairs of points.*

Definition 2.5 *A triangulation is Delaunay if the circle passing through the vertices of each of its triangles does not contain any point of the set in its interior.*

Theorem 2.1 ([16]) *Compass Routing guarantees delivery on Delaunay triangulations.*

Proof. Suppose we are located at vertex a at a given instant of the construction of a route to b, and assume a is not adjacent to b (see Figure 2.8). Let

$\triangle xay$ be a triangular face intersected by the line segment ab. Since $\triangle xay$ is a Delaunay triangle, b is not interior to circle C passing through x, a and y. Then ab intersects xy. We want to prove that whether the option for the next vertex of Compass Routing is x or y, it will be closer to b than a. Since the number of vertices is finite, this will be enough to prove the result.

Let c be the center of C, a' the mirror image of a with respect to line bc. p_1 the intersection of segment ab with circle C and let p be the intersection of segment bc with C.

Since the length of bp is the distance from b to the circle, we need only to compare the distance from x or y to p with the length of ap. Without loss of generalization, assume x and c are not on the same side of ab.

First case: x is chosen. Since xp is shorter than ap, we have the result.

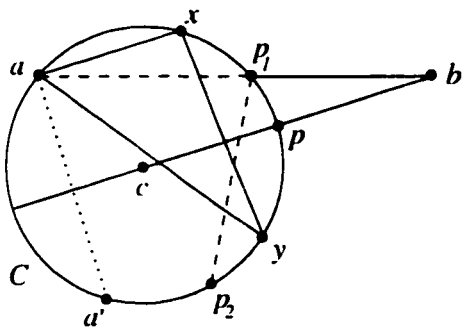


Figure 2.8: Proof of guaranteed delivery for Compass Routing on a Delaunay triangulation

Second case: y is chosen, then $\measuredangle yab < \measuredangle bax$. Construct point p_2 on c such that the length of ap_1 is the same as the length of p_1p_2. We know that $\measuredangle bax < \measuredangle p_1ya$ (x is in the arc $\widehat{p_1a}$) and $\measuredangle p_1ya = \measuredangle p_2ab$ (inscribed angles of arcs circumscribing same length chords). Therefore $\measuredangle yab < \measuredangle p_2ab$, and thus

y is in the arc $\widehat{p_2p_1}$. We only need to ensure that p_2 is in the arc $\widehat{a'a}$, since that will also leave Y in $\widehat{a'a}$ and therefore, closer to b than a. By contradiction, assume p_2 is in the open arc $\widehat{aa'}$, then $\measuredangle p_2ab = \measuredangle p_1p_2a$ (base angles of isosceles triangle $\triangle p_2p_1a$). But $\measuredangle p_2ab > \measuredangle a'ap = \measuredangle pa'a > \measuredangle p_1a'a = \measuredangle p_1p_2a$, which is a contradiction. Therefore y is closer to b than a. ■

In [2] and [17] the authors modified Compass Routing slightly to make it guarantee delivery on graphs whose bounded faces are convex polygons (called *convex subdivisions*). The modification is as follows; choose randomly between v_l or v_r for the next node from v in the route from a to b, where v_l and v_r have the property that $\measuredangle bvv_l$ and $\measuredangle bvv_r$ are the minimum angles among all clockwise and counterclockwise angles from v. They proved that this modification, called *Random Compass*, guarantees delivery on any convex subdivision. They even showed that the route thus generated is not too long compared with the shortest path. With this modification, the routing algorithm on convex subdivisions proceeds as follows:

Algorithm 6. Compass Routing on convex subdivisions

1. Traverse the edges of f, the first face intersected by line segment ab, until we reach the edge $e = uv$ in which this intersection takes place. At this point we can construct the partial path from a to v.

2. Now let f be the next face intersected by ab, the one adjacent to e.

3. Repeat previous steps until b is reached.

•

Assume G is a connected[2], planar geometric graph. G is not necessarily a convex subdivision. In [16] the authors show another modification of Compass Routing which proves that a local routing algorithm can be constructed, guaranteeing delivery on G.

Compass Routing can be applied to *ad hoc* wireless networks. Here antenna positions define the set of vertices of the network and links are produced between antennas located within a certain range given by their *transmission power*. More specifically, suppose antennas are located in a plane terrain. Antenna v_i's *dominance region* (the largest region in which it can receive/transmit a signal) is a circle centered on the antenna with radius r_i, the antenna's transmission power. Vertices v_i and v_j define a communication link (an edge) of the network if $d(v_i, v_j) \leq \min\{r_i, r_j\}$, where d is a distance function.

In [3] and [17] the authors prove that there exists a local routing algorithm guaranteeing delivery for same-power wireless communication networks. They use techniques such as *Gabriel's graph* and the Compass Routing Algorithm for not necessarily convex subdivisions in [16].

In [18] the authors take a similar approach, where a robot may not have a map of the network that it wants to navigate.

The development of *wireless communication networks* and *GPS* systems has motivated new routing techniques for them. Such techniques use the geometric information gathered by each node to provide a better routing scheme (see [7, 12, 15, 23, 25]).

[2] A graph is *connected* if a path can be found between any pair of vertices of the graph.

Chapter 3

Traffic load

We now proceed to define perfect routing schemes. The idea here is to show some classes of graphs with routing schemes on them where the traffic is perfect, that is, every edge receives exactly the same number of paths after each site has sent a single message.

We need to formalize some terms from the previous chapters.

Definition 3.1 *A Routing Scheme or Routing System $\mathcal{R}(G, \mathcal{A})$ consists of a graph $G(V, E)$ with vertex set V (a.k.a. sites or nodes) and edge set (a.k.a. links) E. Edges will be considered undirected unless indicated otherwise. $\mathcal{A}$ defined over G is a routing scheme or algorithm that tries to find a path from any vertex a to any other vertex b.*

This path is not necessarily unique; in fact we shall analyze algorithms in which we generate a random path each time we want to move from vertex a to vertex b. $\pi_{a,b}$ is a not necessarily simple path from a to b represented as a chain of vertices $a_1 a_2 \cdots a_k$, with $a = a_1$, $b = a_k$.

Definition 3.2 *An edge e is used by a path $\pi_{a,b}$ from a to b, denoted $e \in \pi_{a,b}$, if $e = a_i a_{i+1}$ for some $i \in \{1, 2, \ldots, k-1\}$.*

Definition 3.3 *We say that $\mathcal{A}$ guarantees delivery if it always finds a path between every pair of vertices.*

Recall that some algorithms, such as Compass Routing, may in some instances fail to reach the destination node b. From here on, we will only consider algorithms $\mathcal{A}$ which guarantee delivery.

Definition 3.4 *The load l_e of edge e is defined as the number of paths using e.*

Each edge may have a *weight* (its *length* for example) denoted $w(e)$. In the rest of this work we will assume that $w(e) = 1$ and thus the length of a path will be its number of edges. Since we will only consider simple paths, we can always define the weight of a path (its *length*) as $w(\pi) = \sum_{e \in \pi} w(e)$.

3.1 Perfect routing schemes

Consider a shortest path $\pi_{a,b}$ between any pair of vertices a and b of an arbitrary graph $G(V, E)$.

Definition 3.5 *The average load $\bar{l}_G$ of the graph G can be defined as:*

$$\bar{l}_G = \frac{\sum_{a,b \in V} w(\pi_{a,b})}{|E|}$$

Definition 3.6 *A routing scheme is perfect if after it finds a path between each pair of nodes, the number of paths passing through any edge e of $G(V, E)$ is exactly $\bar{l}_G$.*

As we will soon show, some important classes of graphs such as hypercubes have perfect routing schemes. Most networks, however do not have such routing schemes. For example, the network shown in Figure 3.1 has the disadvantage that any routing scheme using shortest paths will overload edge pq. In this particular problem, to avoid congesting edge pq each time we cross from the left to the right part of the network, choose randomly any of the edges connecting p to $x_1, \ldots, x_m$. Then the load at edges px_i and x_iq is reduced by a factor of m. This leaves an open and indeed, as far as we know, unexplored problem of trying to find routing schemes that minimize (or approximate) the traffic load that each edge receives, using paths that are not necessarily the shortest.

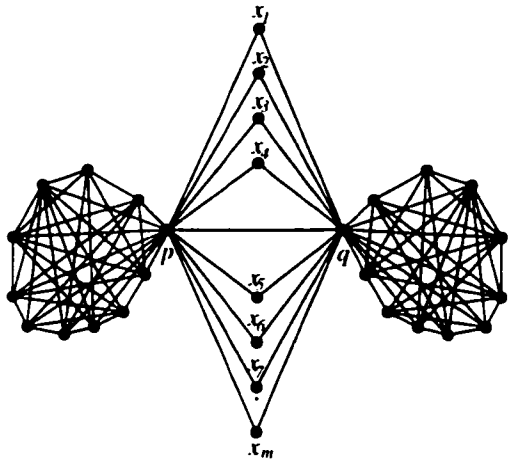


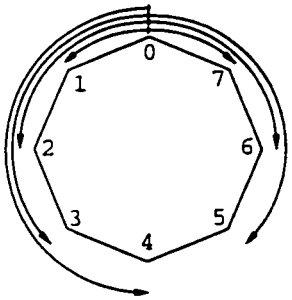
Figure 3.1: It is possible to reduce the (maximum) load by not using shortest paths

In this section we will show some examples of perfect schemes on cycles, hypercubes, 2×2 and 3×3 integer lattices and squared tori. The next section

shows that large lattices cannot afford perfect routing schemes.

Example 3.1. (Cycles)

Consider the routing scheme $\mathcal{R}(\mathcal{C}_n, \mathcal{A})$ where $\mathcal{C}_n$ is the *cycle* or *ring* whose set of vertices is $\{a_0, a_1, \ldots, a_{n-1}\}$ and whose set of edges is $\{a_i a_{i+1} \mid i = 0, 1, \ldots, n-1\}$. Index addition is taken modulo n.



$\mathcal{A}$

Figure 3.2: Routing scheme on a cycle ($\mathcal{C}_8$)

Let the routing scheme $\mathcal{A}$ (see Figure 3.2) be generated by the following algorithm:

Algorithm 7. Cycle routing $(a.b)$

1. Let $i, j \in \{0, 1, \ldots, n-1\}$ such that $a = a_i$, $b = a_j$; $\pi \leftarrow \{a\}$
2. While $i \neq j$
   If $(j - i) \mod n \leq (i - j) \mod n$ then $i \leftarrow (i + 1) \mod n$
   else $i \leftarrow (i - 1) \mod n$
   Add a_i to π

•

That is, $\mathcal{A}$ generates shortest paths from a_i to every other vertex.

For each $i \in \{0, 1, \ldots, n-1\}$, the sum of path lengths is

$$\sum_{k=1}^{n-1} w(\pi_{a_i,a_{i+k}}) = \sum_{k=1}^{\lfloor n/2 \rfloor} k + \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k = \left\lfloor \frac{n}{2} \right\rfloor^2$$

Therefore the average load is $\bar{l}_{\mathcal{R}} = \lfloor n/2 \rfloor^2$.

Since the load for any edge $a_i a_{i+1}$ is also

$$l_{a_i a_{i+1}} = \sum_{k=1}^{\lfloor n/2 \rfloor} k + \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k$$

this scheme is perfect. o

Example 3.2. (Hypercubes and dimension walking)

We defined hypercubes and dimension walking earlier (see Section 2.2). Here we prove that this routing scheme is perfect. Let $\mathcal{R}_d$ be the routing scheme for dimension walking over hypercube $\mathcal{Q}_d$ (Algorithm 2).

In order to calculate the average load for $\mathcal{R}_d$, observe that vertex A's neighbors at distance 1 (joined to a by a path of length 1) have indices differing from a's by one bit. Similarly, indices of vertices at distance t from a differ from the index of a in exactly t bits and there are $\binom{d}{t}$ of them. Therefore, the sum of the lengths of paths at distance t is $t\binom{d}{t}$.

Now, since each of the 2^d vertices in $\mathcal{Q}_d$ has d neighbors, the number of edges in the hypercube is $d2^{d-1}$. Thus the average load for $\mathcal{Q}_d$ is

$$\bar{l}_{\mathcal{Q}_d} = \frac{2^d \sum_{t=1}^{d} t\binom{d}{t}}{d2^{d-1}} = 2^d$$

Theorem 3.1 *Routing scheme $\mathcal{R}_d$ (dimension walking in the hypercube) is perfect.*

Proof. By induction over d, there are exactly 2 paths for the segment ab representing $\mathcal{Q}_1$: one from a to b and one from b to a. Assume the result holds for any hypercube in $d-1$ dimensions. Divide $\mathcal{Q}_d$ into two $d-1$ dimensional hypercubes. Those vertices whose index binary representation has the leftmost bit set to 0 will be in $\mathcal{Q}_d^0$ and the rest, that is, those whose index has the leftmost bit set to 1 will be in $\mathcal{Q}_d^1$. Call any edge joining vertices from both hypercubes a *bridge edge* (see Figure 3.3).

By the induction hypothesis, routes between every pair of vertices of $\mathcal{Q}_d^0$ (respectively $\mathcal{Q}_d^1$) produce a load in each edge of $\mathcal{Q}_d^0$ ($\mathcal{Q}_d^1$) of 2^{d-1}. Note that those routes only use edges of $\mathcal{Q}_d^0$ ($\mathcal{Q}_d^1$). This is because crossing the bridge would imply a modification of the leftmost bit on the index of the current vertex in the path.

Let a_i be a vertex of $\mathcal{Q}_d^0$, for $i \in \{0, 1, \ldots, 2^{d-1}-1\}$. Then $a_{i+2^{d-1}}$ is the vertex of $\mathcal{Q}_d^1$ at the other end of the bridge edge joining a_i and $a_{i+2^{d-1}}$. Observe that any bridge edge $a_i a_{i+2^{d-1}}$ can only be used as a final edge for any of the 2^{d-1} paths (one per vertex) originating at vertices of $\mathcal{Q}_d^0$ ($\mathcal{Q}_d^1$ respectively). This is because dimension walking can only modify the leftmost bit at the end of a path. Thus any bridge edge $a_i a_{i+2^{d-1}}$ is used $2^{d-1}+2^{d-1}=2^d$ times.

Note also that the 2^{d-1} paths crossing from $\mathcal{Q}_d^0$ to $\mathcal{Q}_d^1$ ($\mathcal{Q}_d^1$ to $\mathcal{Q}_d^0$ resp.) using a_i ($a_{i+2^{d-1}}$), for each $a_i \in \mathcal{Q}_d^0$ ($a_{i+2^{d-1}} \in \mathcal{Q}_d^1$) define a set of routes between each pair of vertices of $\mathcal{Q}_d^0$ ($\mathcal{Q}_d^1$) plus the bridge edges, which by induction hypothesis add an additional load of 2^{d-1} to each edge of $\mathcal{Q}_d^0$ ($\mathcal{Q}_d^1$). Thus these edges also receive a total load of 2^d. ■

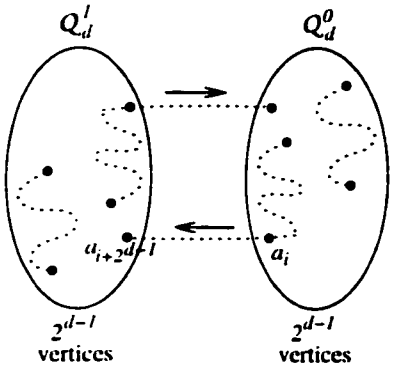


Figure 3.3: Dimension walking on the hypercube $\mathcal{Q}_d$ is a perfect scheme.

In a similar way we could prove that all the other $d! - 1$ variations of dimension walking over $\mathcal{Q}_d$ are also perfect.

As a practical application of hypercubes, in [28] the authors show load balancing on multiprocessor computers using hypercube topology. They assume that each vertex is a processor and that they can send messages at the same time. o

The next example, already seen as the hypercube $\mathcal{Q}_2$, will be used to introduce a notation for routes on the integer lattice for the following examples.

Example 3.3. ($\mathcal{L}_2$)

For the integer lattice $\mathcal{L}_2$, let "r", "u", "d", and "l" denote the right, up, down and left neighbors of a vertex respectively, if they exist. Consider first the vertex $a = (0,0)$ as the origin for routes. To reach every other vertex from A, we will use concatenations of these directions. For example "r", "u", and "ru" describe routes from a to $(1,0)$, $(0,1)$, and $(1,1)$ respectively (see Figure 3.4a).

Routes corresponding to Figure 3.4a		
origin	route descriptions	rotations
$(0,0)$	r, u, ru	3

If we consider $(0,0)$ as the origin vertex and we rotate the whole diagram and its route descriptions 90°, 180°, and 270°, we will have a set of routes between every pair of vertices.

For example, considering the three rotations, ru translates not only into path $(0,0)(1,0)(1,1)$, but also into $(1,0)(1,1)(0,1)$, $(1,1)(0,1)(0,0)$ and $(0,1)(0,0)(1,0)$. That is, rotating 90° translates the meaning of the directions as follows: $r \hookrightarrow u$, $u \hookrightarrow l$, $l \hookrightarrow d$ and $d \hookrightarrow r$. Rotating 180° translates the meaning of directions as follows: $r \hookrightarrow l$, $u \hookrightarrow d$, $l \hookrightarrow r$ and $d \hookrightarrow u$. Rotating 270° translates the meaning of directions as follows: $r \hookrightarrow d$, $u \hookrightarrow r$, $l \hookrightarrow u$ and $d \hookrightarrow l$.

We can observe that this set of routes define a perfect scheme since four routes use every single edge. In fact, the set of routes is the same as that generated by the dimension walking algorithm over $\mathcal{Q}_2$. ∘

Example 3.4. ($\mathcal{L}_3$)

Going a step further, we now consider the integer lattice $\mathcal{L}_3$.

We can define at least two perfect routing schemes. To describe them, we will use the notation of the previous example:

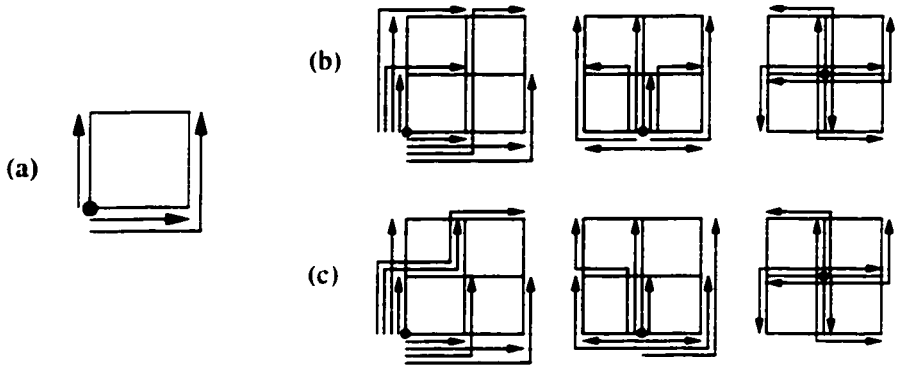


Figure 3.4: Routing schemes on $\mathcal{L}_2$ and $\mathcal{L}_3$

Routes corresponding to Figure 3.4b		
origin	route descriptions	rotations
(0,0)	$u, r, uu, rr, ur, uur, rru, ruur$	3
(1,0)	$u, r, l, uu, ul, luu, ur, ruu$	3
(1,1)	$r, u, d, l, ld, ul, dr, ru$	0

As before, path description $ruur$ translates, for example, into paths

$$(0,0)(1,0)(1,1)(1,2)(2,2), (2,0)(2,1)(1,1)(0,1)(0,2)$$

$$(2,2)(1,2)(1,1)(1,0)(0,0) \text{ and } (0,2)(0,1)(1,1)(2,1)(2,0)$$

The other perfect scheme is:

Routes corresponding to Figure 3.4c		
origin	route descriptions	rotations
(0,0)	$u, r, uu, rr, ru, uru, rru, urur$	3
(1,0)	$l, r, u, uu, lu, ulu, ru, ruu$	3
(1,1)	$r, u, d, l, ld, ul, dr, ru$	0

In the next section we will calculate the average load on an integer lattice. For now the interested reader can verify in these small examples that every edge receives exactly the same traffic load. o

Example 3.5. (Squared torus)

The *squared torus* is a network which supports many deterministic and perfect routing schemes. Here the set of vertices is

$$V = \{(i,j) \mid i,j \in \{0,1,\ldots,n-1\}\}$$

and the set of edges is

$$E = \{(i,j)(i+t,j+u) \mid t,u \in \{-1,0,1\}, |t+u| = 1\}$$

(sums are taken modulo n).

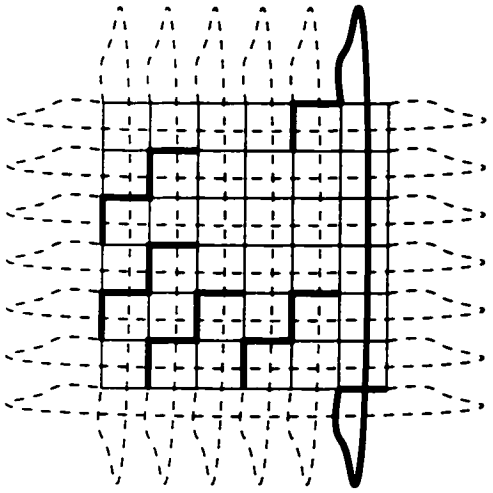


Figure 3.5: Squared torus and replicated paths

Every node has four neighbors and because of its symmetry, the route from (a,b) to (i,j) can be replicated to build the route from $(a+x,\ b+y)$ to $(i+x,\ j+y)$ where $x,y \in \{0,1,\ldots,n-1\}$ (see Figure 3.5).

Actually, while this result can be generalized to the Cartesian product of cycles with the same length $\overbrace{\mathcal{C}_n \times \mathcal{C}_n \times \cdots \times \mathcal{C}_n}^{k}$, it is not true for the product of different length cycles $\mathcal{C}_{n_1} \times \mathcal{C}_{n_2} \times \cdots \times \mathcal{C}_{n_k}$.

Algorithm 8. Routing on $\prod_{j=0}^{k-1} \mathcal{C}_n$

1. Let $i_0, i_1, \ldots, i_{k-1}, j_0, j_1, \ldots, j_{k-1} \in \{0, 1, \ldots, n-1\}$ so that $a = (i_0, i_1, \ldots, i_{k-1})$ and $b = (j_0, j_1, \ldots, j_{k-1})$, $\pi \leftarrow \{a\}$

2. For $r = 0..k-1$

   If $(j_r - i_r) \mod n \leq (i_r - j_r) \mod n$ then $i_r \leftarrow (i_r + 1) \mod n$

   else $i_r \leftarrow (i_r - 1) \mod n$

   Add $a = (i_0, i_1, \ldots, i_{k-1})$ to π

• ○

3.2 Topological influence

We note however, that some topological properties of the graph affect the load independently of the algorithm.

Consider, for example, the symmetry of cycles. In Example 3.1 we saw an algorithm over cycles in which n vertices reach average load. A way of breaking the symmetry of a cycle would be to delete an edge from it. In this case we obtain the *String* $\mathcal{S}_n$ with n vertices. We say that $\mathcal{S}_n$ is not symmetric in the topological sense, because nodes at both ends have only one neighbor, while the other nodes have two neighbors. How does lack of topological symmetry affect the load?

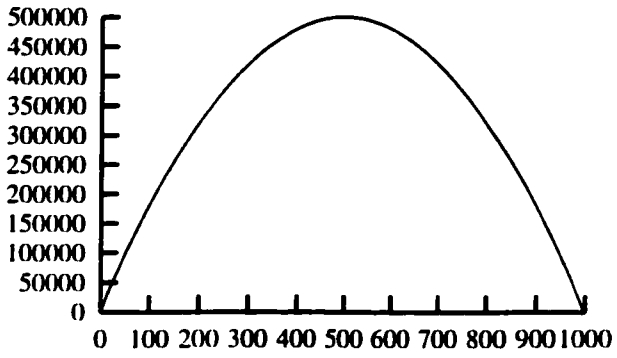


Figure 3.6: Load curve of $\mathcal{S}_{1000}$

We chose this example because of its simplicity and because there is only one algorithm to route in the string. If the string $\mathcal{S}_n$ has vertices $\{v_0, v_1, \ldots, v_{n-1}\}$ and edges $\{e_i = v_i v_{i-1} \mid i = 1, 2, \ldots, n-1\}$, then edge e_1 is used in paths originating from v_0 to all the other $n-1$ vertices. It is also used on paths originating from all other vertices and destination v_0. In this way $l_{e_1} = 2(n-1)$. Edge e_2 is used in paths originating from both v_0 and v_1 to all other vertices as well as in paths originating from all other vertices and directed to them, that is $l_{e_2} = 2 \cdot 2(n-2)$. In general, edge i is used by paths originating from $v_0, v_1, \ldots, v_{i-1}$, directed to the other $n-i$ vertices, and also on paths originating from the other $n-i$ vertices, and directed to them, that is $l_{e_i} = 2i(n-i)$. Observe the disproportionate traffic in Figure 3.6 when routing between each pair of vertices of S_{1000}. Central edges may reach the order of half a million routes passing through them, while edges at both ends receive 1998 routes, which is comparatively insignificant. A perfect routing algorithm over the string is of course out of question.

We can also observe this topological influence over the load produced by

algorithms on the integer lattice.

3.3 Imperfect routing schemes

Starting from $n = 4$, it is not possible to reach the average load (of 27) at each edge of the integer lattice $\mathcal{L}_n$. Trial and error shows that there is always an edge with load of at least 32.

This leads us to wonder if that is the case for a larger integer lattice. Indeed we found the following result which in particular shows that whenever a subset of vertices of a network determine an integer lattice larger than 3×3, the network cannot have a perfect routing scheme.

Theorem 3.2 *For any routing scheme in the $n \times n$ lattice, and $n \geq 4$, there is always an edge with an above-average load.*

Proof. Assume n even and divide the lattice into four sub-lattices, each of size $n/2 \times n/2$. Call the set of edges joining adjacent sub-lattices (see Figure 3.7a) a *bridge*. Shortest routes between each of the four pairs of adjacent sub-lattices use the bridge $n^4/8$ times. Also, shortest routes between each of the two pairs of opposite sub-lattices use the bridge $n^4/4$ times. Since the number of edges in the bridge is $2n$, we have to use an edge b on the bridge at least

$$\frac{4\left(\frac{n^4}{8}\right) + 2\left(\frac{n^4}{4}\right)}{2n} = \frac{n^3}{2}$$

times. This confirms that the bridge edges for $\mathcal{L}_{4\times4}$ have load of 32 as in our example. If n were odd, the value would be higher.

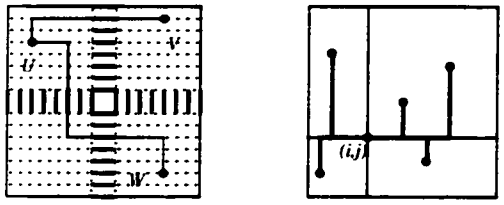


Figure 3.7: (a) Adjacent $U - V$ and opposite $U - W$ sub-lattices determined by the bridge in bold lines, (b) Sub-lattices determined by (i, j)

To obtain the sum of the lengths of the shortest routes between every pair of vertices, we fix an origin vertex (i, j). Then we obtain the partial sums of route lengths from (i, j) to vertices in the four sub-lattices determined by (i, j). A systematic way to achieve this is by using the Greedy Algorithm as in Figure 3.7b. Thus we obtain:

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\left(\sum_{r=0}^{m-i}\left(\frac{(n-j+1)(n-j)}{2}+(n-j+1)r\right)\right.$$
$$+\sum_{r=1}^{i-1}\left(\frac{(n-j+1)(n-j)}{2}+(n-j+1)r\right)$$
$$+\sum_{r=0}^{i-1}\left(\frac{j(j-1)}{2}+(j-1)r\right)$$
$$\left.+\sum_{r=1}^{m-i}\left(\frac{j(j-1)}{2}+(j-1)r\right)\right)$$

which simplifies to $\frac{2}{3}n^5 - \frac{2}{3}n^3$. Since the number of edges in the lattice is $e = 2n^2 - 2n$, we get an average of

$$\frac{n^3+n^2}{3}.$$

In this way, bridge edge b has a larger than average load, for $n \geq 4$. ■

3.4 Comments

A perfect scheme produces the same traffic load on every edge while generating one path between every directed pair of nodes. It is not difficult to see that if multiple paths are described between random pairs of nodes on the same network by the perfect scheme, after some time the traffic will start distributing evenly. This alone is a good reason why some multiprocessor computers are or can be built on networks having perfect routing schemes.

Such networks include the torus, hypercube of interconnected multiprocessor computers, ring-like Local Area Networks, etc. However, given that these may be very restrictive shapes for a network, a proposal may be not to build the entire network, say hypercube shaped, but to have an important and highly communicated subnetwork with that shape, the *backbone*, for which we need perfectly distributed traffic. The backbone would communicate clusters of local sites. Then use two levels of routing schemes, one internal to each cluster and one for routing over the backbone. The development of such strategies would be an interesting topic for further research.

Given that a perfect routing scheme does not always exist, we will say that a good routing scheme is one approaching average load on each edge. In Chapter 4 we will slightly modify the definition of perfect schemes, allowing more than one scheme to coexist in the graph in order to reach the same load on every edge.

Chapter 4

Edge-transitive graphs

As mentioned in the previous chapter, finding networks for which a perfect routing scheme exists is extremely difficult. As we will now see, even graphs which are highly symmetric may not have such routing schemes. For example the icosahedron has no such scheme, as will be proved shortly. Intuitively, this seems highly unsatisfactory. A natural question can be: is it possible to somehow modify our definitions so that in the long run, graphs such as the icosahedron have a perfect routing scheme? Our goal in this chapter is to prove that indeed, it is possible not only for the icosahedron, but in general for edge transitive graphs.

4.1 Routing in the platonic solids

Consider for example, the case of the graph $\mathcal{I}$ obtained from the vertices and edges of the icosahedron (see a planar embedding in Figure 4.1). Suppose the 20 triangular faces, 12 vertices and 30 edges of this graph represent a

communication network for which we want to send one message between each pair of vertices.

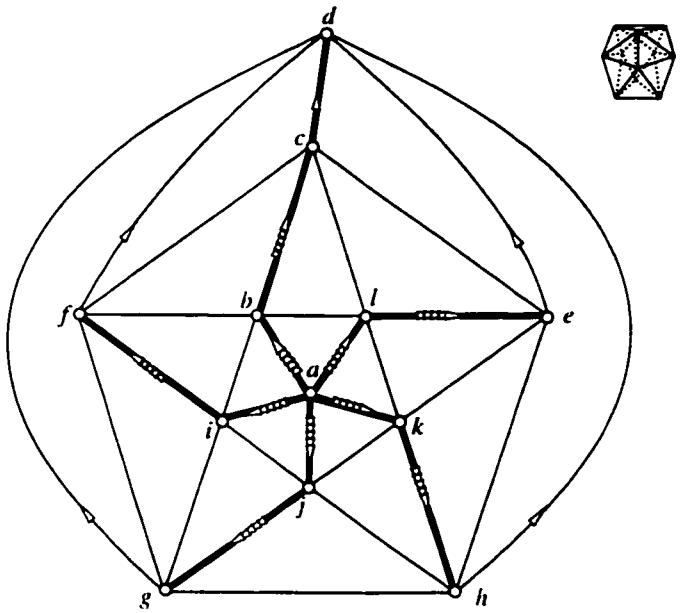


Figure 4.1: Tree (in bold line) rooted at a over the icosahedron

Observe that any vertex a of $\mathcal{I}$ has five vertices at distance one, five at distance two and one at distance three. Thus the average load of any edge of $\mathcal{I}$ is

$$\frac{12 \cdot (1 \cdot 5 + 2 \cdot 5 + 3 \cdot 1)}{30} = 7\frac{1}{5}$$

This tells us that if we choose a fixed set of paths to route on $\mathcal{I}$, some edges will have a traffic load of at least eight, while others at most seven. In other words, traffic will be unevenly distributed, which is unsatisfactory given the high degree of symmetry of $\mathcal{I}$.

The following question arises naturally: is it possible to define a routing

scheme over $\mathcal{I}$ that will distribute the traffic load evenly on all the edges of $\mathcal{I}$ over a specific period of time?

To accomplish this, we will change our model of routing schemes slightly. Suppose that instead of only one message between each pair of vertices, k messages are sent for some value of k. Is it possible to find a set of paths to route the messages such that traffic is evenly distributed among the edges of $\mathcal{I}$? We show that indeed this is the case.

Consider $\mathcal{I}$ and a fixed embedding of it in the plane, and let $\mathcal{T}_1$ be the shortest path spanning tree of $\mathcal{I}$ shown in bold in Figure 4.1 and assume that we want to send five messages between any pair of vertices. Let $\mathcal{T}_2$, $\mathcal{T}_3$, $\mathcal{T}_4$ and $\mathcal{T}_5$ be the spanning trees of $\mathcal{I}$ obtained from rotating $\mathcal{T}_1$ around a. Thus paths $(abcd, aif, ajg, akh, ale)$ of $\mathcal{T}_1$ become $(aifd, ajg, akh, ale, abc)$ on $\mathcal{T}_2$, $(ajgd, akh, ale, abc, aif)$ on $\mathcal{T}_3$, $(akhd, ale, abc, aif, ajg)$ on $\mathcal{T}_4$, and $(aled, abc, aif, ajg, akh)$ on $\mathcal{T}_5$ respectively.

For each vertex $v \neq a$, let f_v be an automorphism on $\mathcal{I}$ sending a to v and preserving orientation on the faces of the planar embedding of $\mathcal{I}$. For each $\mathcal{T}_i$ and v we will consider in our routing the paths defined by all trees $f_v(\mathcal{T}_i)$.

The routing scheme $\bar{\mathcal{R}}$ will send five messages between each pair of vertices as follows: to route from a to vertex x, send the first message along a path on $\mathcal{T}_1$, send the second message along a path on $\mathcal{T}_2$, and continue in a similar way.

Note that some messages between two vertices will use the same path on $\mathcal{I}$. For example, messages from a to c will use path abc for all five trees, while messages from a to d will use $abcd$ on $\mathcal{T}_1$, $aifd$ on $\mathcal{T}_2$, $ajgd$ on $\mathcal{T}_3$, $akhd$ on

$\mathcal{T}_4$ and *aled* on $\mathcal{T}_5$.

Let ε be an edge in $\mathcal{I}$. ε is the third edge in a path from the root of only two trees, because each of its endpoints is at distance three from exactly one vertex (see Figure 4.2a).

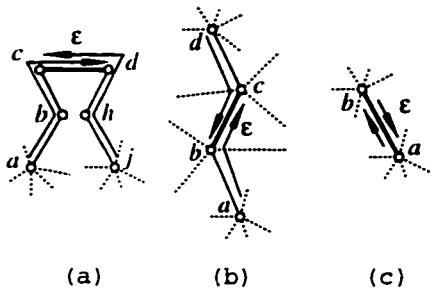


Figure 4.2: Three cases for an edge of the icosahedron

Edge ε is the second in a path from the root of exactly ten trees. To see this, first observe for example edge bc in the direction from b to c. By the definition of our trees, it is a second edge in a path from root x iff there is exactly one neighbor of b in the counterclockwise (right) region of the plane between edges xb and bc. In this case, only vertex a out of all the vertices at distance one from b ($\{c, a, i, f\}$), allows this single neighbor of b. The neighbor is l and the root x has to be a, whose five trees $\mathcal{T}_1, \ldots, \mathcal{T}_5$ have bc as a second edge. Analogously for the direction from c to b we find that the five trees rooted at d also have bc as second edge of a path. Note that if we also count the two paths of length three using the bc as a second edge, those originating in a and d in our example, we have that twelve paths use e as a second edge (see Figure 4.2b).

Finally it is not difficult to see that the five trees at both endpoints of edge ε have ε as first edge of a path from them. The number of paths

using ε as a first edge is the sum of the paths of length three (two in our example, one originating from b and one from c), of length two (ten in our example, five originating at c and five originating at b) and those of length one corresponding to the trees mentioned (ten), see Figure 4.2c. That is, twenty-two paths use ε in this case.

To summarize, ε is third in two 3-length paths, second in ten 2-length paths and is also second in two 3-length paths. ε is first in ten 1-length paths, it is first also in ten 2-length paths and finally first in two 3-length paths.

Thus to calculate the traffic load for edge ε we only need to add up the load for these three cases:

$$l_\varepsilon = 2 + 12 + 22 = 36$$

This corresponds to the expected average load

$$\bar{l}_\mathcal{I} = 5 \cdot 7.2 = 36$$

It is not difficult to see that this procedure can be applied to all the Platonic solids; the tetrahedron, cube or hexahedron, octahedron, dodecahedron and icosahedron, the 3D solids having 4, 6, 8, 12 and 20 isomorphic faces respectively.

Graphs derived from the Platonic solids				
Graph	v	e	v_i	l_e
Tetrahedron	4	6	$v_1 = 3$	2
Cube	8	12	$v_1 = 3$, $v_2 = 3$, $v_3 = 1$	8
Octahedron	6	12	$v_1 = 4$, $v_2 = 1$	3
Dodecahedron	20	30	$v_1 = 3$, $v_2 = 6$, $v_3 = 6$, $v_4 = 3$, $v_5 = 1$	$33\frac{1}{3}$
Icosahedron	12	30	$v_1 = 5$, $v_2 = 5$, $v_3 = 1$	$7\frac{1}{5}$

In this table, the graphs derived from the corresponding platonic solids have v vertices, e edges, v_i vertices at distance i from a fixed vertex, and have average load per edge l_e, which can be calculated as $\frac{v\sum_{i=1}^{k} iv_i}{e}$.

However the only interesting case besides the icosahedron is the dodecahedron, since we have seen that the cube has a perfect routing scheme (dimension walking) and integer average loads for the tetrahedron and octahedron give us a hint that a perfect routing is possible. Indeed Figure 4.3 shows perfect routing schemes for these two Platonic solids.

For the dodecahedron we need to build a shortest path spanning tree $\mathcal{T}_1$ like the one in Figure 4.4 and its two rotations around vertex a; $\mathcal{T}_2$ and $\mathcal{T}_3$. Now, as with the icosahedron, to route from vertex a to vertex x, three

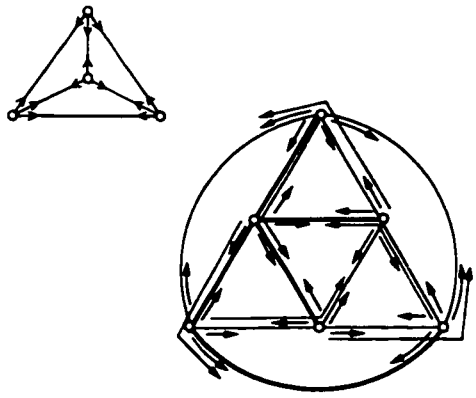


Figure 4.3: Perfect routing schemes over the tetrahedron and the octahedron

messages are sent: one along a path in $\mathcal{T}_1$, another along a path in $\mathcal{T}_2$ and the third along a path in $\mathcal{T}_3$.

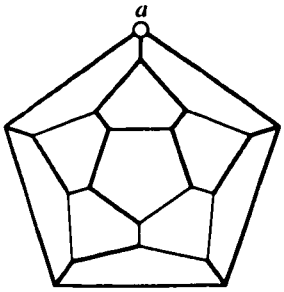


Figure 4.4: A shortest path spanning tree on the dodecahedron

4.2 Almost perfect routing

We now show that a similar approach can be obtained over *edge-transitive graphs*, which are a generalization of the Platonic solids.

Definition 4.1 *An edge-transitive graph has any two edges equivalent under*

an automorphism. That is, edge e' is equivalent to e under automorphism f if $e' = f(e)$.

Formally speaking, we have been applying automorphisms of icosahedron $\mathcal{I}$ over tree $\mathcal{T}_1$ to obtain its rotations and their mapping onto every other vertex of the graph.

Definition 4.2 *An automorphism of a graph G is a 1-1 function $f : G \longrightarrow G$ preserving incidence.*

Definition 4.3 *If edges e_1 and e_2 have a common vertex v, and edges $f(e_1)$ and $f(e_2)$ have common vertex $f(v)$ then function f is said to preserve incidence.*

Let $F_v(G)$ be the set of automorphisms of the edge-transitive graph G such that $f \in F_v(G)$ if $f(r) = v$, and $\mathcal{T}$ a shortest path spanning tree of G rooted at r. To route from vertex v to vertex x, our routing scheme $\bar{\mathcal{R}}(G, \mathcal{T})$ sends a message along the path of the tree $f(\mathcal{T})$ joining them, for each $f \in F_v(G)$. Thus $\bar{\mathcal{R}}$ will send $|F_v(G)|$ messages from vertex v to vertex x.

It is not difficult to see that if $g \in F_x(G)$ and f is an automorphism such that $f(x) = y$, then $f \circ g \in F_y(G)$. In this way we have that $|F_x(G)| = |F_y(G)|$ for any vertices x and y of G.

Now let e and e' be two edges of G incident at vertex w and π be a path obtained for our routing scheme $\bar{\mathcal{R}}(G, \mathcal{T})$ using e (see Figure 4.5). By definition there exists a vertex v and an automorphism $g \in F_v(G)$ such that $\pi = g(\pi_0)$ for some path π_0 in $\mathcal{T}$. Also by the definition of edge-transitive,

there exists an automorphism f such that $e' = f(e)$. Then automorphism $f \circ g$ sends π_0 into a path using edge e'. Similarly, whenever we have a path using e' we can find a valid path for the scheme using e. Thus the load of edges incident at w is the same, but by transitivity and since G is connected, this result implies that every edge of G has the same traffic load.

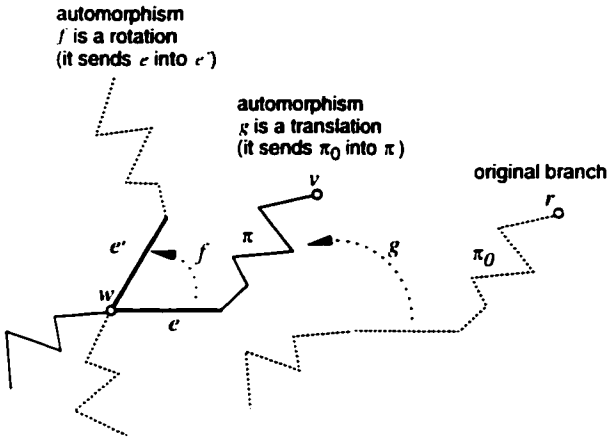


Figure 4.5: Edge-transitive graphs have a routing scheme evenly distributing the traffic on each edge

Theorem 4.1 *If G is an edge-transitive graph and $\mathcal{T}$ any shortest path spanning tree on G, then the routing scheme $\bar{\mathcal{R}}(G, \mathcal{T})$ produces the same traffic load on every edge of G by sending $|F_v(G)|$ messages from each vertex v to every other vertex x in the graph.*

This approach is somewhat of a detour from local routing, since each message has to carry information (say, a number $l \in \{1, 2, \ldots, |F_v(G)|\}$) identifying the automorphism following it. Note however that this routing achieves a perfect routing over time. In many applications the tradeoffs that

this approach represents are worth their cost, as congestion in communication networks is a major problem whose solution is hard to achieve. Moreover this is common practice in applications such as Internet, where each node stores a number ($|F_v(G)|$ in our case) of *routing tables*, which can be consulted by an algorithm according to the label in the message.

Chapter 5

Uniform route selection

In the previous chapter we saw that for certain graphs it is not always possible to find a route between every pair of vertices such that every edge has the same load.

The single-minded approach of deterministically assigning a single path to every pair of nodes can often lead to overloading of some edges, creating traffic problems. This effect is more dramatic over certain graphs such as the integer lattice, where there is no way of lowering the traffic over specific edges, in this case, bridge edges.

In the previous chapter we presented a method that alternates routing schemes for every message sent by each site, in order to reach the same traffic load in each edge after a few messages are sent. We also saw that this method works for edge-transitive graphs. But while searching for the best that can be done in general graphs, another approach was considered. The idea is to show how can we improve the traffic of messages between only two sites by using some sort of randomization, in the hope that this restriction improves

overall traffic, while still trying to meet the conditions stated in Section 1.3.

So instead of assigning a fixed path to route from a to b for every pair of nodes a and b of a network, our approach now is to choose a random path which complies with at least two conditions:

- Paths are chosen uniformly
- Only local decisions are taken at each step of the scheme.

By *uniform* we mean that given the two sites, every shortest path from a to b has an equal probability of being selected. By *local* we understand Definition 1.2.

Briefly analyzing these two conditions, we observe that to achieve uniformity with a traditional approach we would need a centralized entity to know/calculate all the shortest paths from a to b and choose one of them randomly. But this contradicts the local condition, not to mention that for practical implementations, this task would require an enormous amount of memory to store all routes in order to be able to pick one of them randomly. We need a scheme, if possible, to meet both conditions at the same time.

5.1 Biased routes

Before describing our solution, we need to observe that not all random routing schemes choose paths uniformly. We exemplify this with a commonly used routing algorithm on integer lattices:

Algorithm 9. 50–50 routing (a,b)
Chooses with equal probability the horizontal or the vertical edge leading to

a vertex closer to b.

1. Let $a = (a_x, a_y)$, $b = (b_x, b_y)$, $\pi \leftarrow \{a\}$,
   $\sigma_x \leftarrow \text{sign}(b_x - a_x)$, $\sigma_y \leftarrow \text{sign}(b_y - a_y)$
2. While $a_x \neq b_x$ and $a_y \neq b_y$
   (a) Let t be a Bernoulli(1/2)[1] random variable
   (b) If $t = 1$ then let $a_x \leftarrow a_x + \sigma_x$
   else let $a_y \leftarrow a_y + \sigma_y$
3. While $a_x \neq b_x$ let $a_x \leftarrow a_x + \sigma_x$
4. While $a_y \neq b_y$ let $a_y \leftarrow a_y + \sigma_y$

•

It is worth mentioning that 50–50 can be generalized to arbitrary networks. The idea is that while routing from node a to node b, the next node to visit is chosen as follows: let $s = a$ be the initial position. If k edges incident to s belong to shortest paths from s to b (obtained, say, using a Dijkstra's-like algorithm), choose any one of them with equal probability, and traverse it. The end point reached will be the new vertex s; this procedure will be repeated until b is reached.

The algorithm chooses paths in a biased way, and even worse, while some edges are underused, others tend to be overused. To observe this, consider the 50–50 algorithm on the lattice $\mathcal{L}_{10\times 1}$ (see Figure 5.1).

Let Π be the set of all different shortest paths from vertex a to vertex b generated by any routing scheme. Observe that there is only one route in Π

[1] Bernoulli(p) random variables have a value 1 with probability p and 0 otherwise.

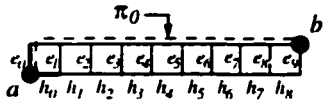


Figure 5.1: Path π_0 is used half the times, while other paths are underused

---

using e_0 (the path π_0), while there are nine routes using h_0 (the paths using one of the vertical edges $e_1, e_2, \ldots, e_9$).

Now suppose that n not necessarily different paths are generated from a to b using 50–50 algorithm.

In this experiment, path π_0 (that is, the path using e_0) will be generated approximately $n/2$ times, the path using e_1 will be generated approximately $n/4$ times, the path using e_2 will be generated approximately $n/8$ times, ..., the path using e_8 will be generated approximately $n/2^9$ times and the path using e_9 will be generated approximately $n/2^{10}$ times on average.

Observe that while path π_0 will be generated almost half of the times, the path using e_9 (the one using edges $h_1h_2h_3h_4h_5h_6h_7h_8e_9$) is almost never generated during this experiment. This kind of disproportion or *bias* in route utilization and on edge loads (see Figure 5.2a) is unacceptable for practical purposes.

We now present an algorithm which will perform better than 50–50 in terms of distributing the paths between each pair of sites uniformly while at the same time improving edge utilization in this lattice.

5.2 Uniform randomized routing algorithm

The objective is to develop an algorithm that compensates for disproportion in the use of routes, while still taking local decisions.

In order to route from a to b, the idea is to have a preprocessing step to set the probability of using an edge e coming out of a (or any other intermediate site s) according to the number of shortest paths reaching b and passing through e. Then the main algorithm proceeds from site to site as for the 50–50 algorithm, but instead of choosing the next edge to visit with equal probability, it uses the probabilities set at the preprocessing step.

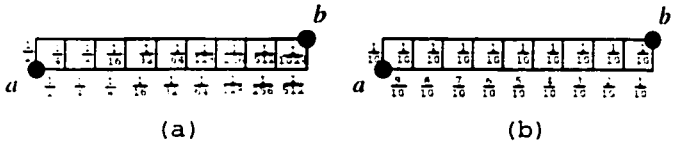


Figure 5.2: Probabilities of using each edge in (a) 50–50, and (b) uniform randomized routing algorithms, for a path from a to b

Continuing with our example, in network $\mathcal{L}_{10\times1}$, to route from a to b (see Figure 5.2b), since there are 9 paths using h_0 compared to one using c_0 (recall Figure 5.1), we should give a higher probability (90%) to starting a path using h_0. Since we are in an integer lattice, these probabilities can be readily calculated based on current position, and the destination coordinates as will be shown in Section 5.4. We will shortly prove that the routes from a to b are chosen uniformly.

Furthermore, in this example the probability of using e_9 is 1/10, compared to 1/1024 for the 50–50 algorithm. It could be argued that the probabilities

are still disproportionate but there is no way of circumventing this, as proved by Theorem 3.2.

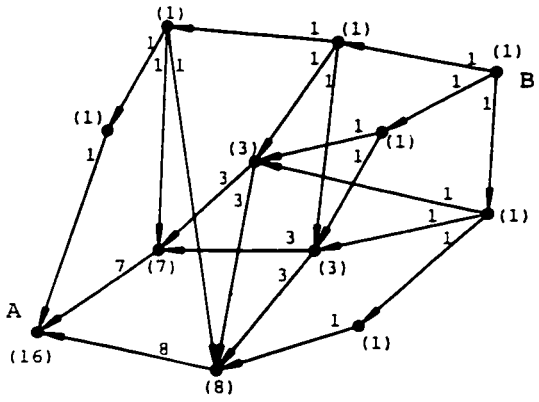


Figure 5.3: Graph showing c_b and (m_b) for each vertex s and the edges of directed tree $\mathcal{T}_b$, for a fixed destination b

We now proceed to describe the algorithm to preprocess any graph $G(V, E)$ so that for every pair of vertices a and b and any edge $e = au$ in the graph, we know how many shortest routes from a to b use e.

Preprocess 2. (Calculate c_b and m_b) Calculate the number $c_b[e]$ of paths originating at b using e as well as the number $m_b[s]$ of routes reaching s from b.

0. For each node b in the graph:

   (0.1) Run the *Breadth First Search* algorithm starting at b, yielding a shortest route directed tree $\mathcal{T}_b$ with b as root.

   We will use a standard broadcast technique (also called flooding) to determine $c_b[e]$ (see Figure 5.3).

At each node s of $\mathcal{T}_b$, keep the auxiliary counter $m_b[s]$. Initially $m_b[s] \leftarrow 0\ \forall s \in V\backslash\{b\}$, $m_b[b] \leftarrow 1$, $c_b[e] \leftarrow 0\ \forall e$.

(0.2) For each vertex s of $\mathcal{T}_b$, traversed by the Breadth First Search starting from b. Let $N[s]$ be the set of neighbors of s.

For every $t \in N[s]$

If st is an edge of $\mathcal{T}_b$

Set $c_b[st] \leftarrow m_b[s]$ and $m_b[t] \leftarrow m_b[t] + c_b[st]$

•

This preprocessing uses *Breadth First Search* to obtain only shortest paths from each vertex s to vertex b. Other shortest route methods may possibly replace Step 0.1 for specific networks. The preprocessing is simply a count of the shortest paths traversing the branches defined by BFS.

Observe that the above preprocessing enables the following algorithm not only to route from a to b, but also from any other vertex s to b.

Algorithm 10. Uniform Randomized Routing(a,b)

Builds π, a uniformly chosen path from a to b.

1. $\pi \leftarrow \{a\}$, $s \leftarrow a$,

2. Let x be a Uniform[1, $m_b[s]$] random variable[2]. Assume $N[s] = \{t_1, \ldots, t_k\}$ and edge $e_j = st_j$, for $j = 1, \ldots, k$. Assume also that $c_b[e_0] = 0$ and let i be the smallest index such that

$$x \in \left(\sum_{j=0}^{i-1} c_b[e_j] \, , \sum_{j=0}^{i} c_b[e_j] \right].$$

[2] x is a Uniform[s, t] random variable if probability $\mathbf{P}(x = x_0) = 1/(t - s + 1)$, for $x_0 \in [s, t] \cap \mathbf{Z}$ and 0 otherwise.

3. Add t_i to π, $s \leftarrow t_i$, and go to Step 2 if b has not been reached.

•

This algorithm produces π, a shortest path between a and b. We will prove now that π is indeed uniformly distributed among all shortest routes from a to b.

For example, in Figure 5.3 to route from A to B initially $\pi \leftarrow \{A\}$ and $c_b[e_1] = 8$, $c_b[e_2] = 7$, $c_b[e_3] = 1$. Intervals are $(0, 8]$, $(8, 15]$ and $(15, 16]$. If for example $x \leftarrow 10$ the algorithm moves from A using the edge with 7 paths.

Theorem 5.1 *Let π be a random route obtained from Algorithm 10 to route from a to b along the vertices of graph $G = (V, E)$. Let π_0 be a fixed shortest path between a and b. Then*

$$\mathbf{P}(\pi = \pi_0) = \frac{1}{m_b[a]},$$

where $m_b[a]$ is the number of shortest paths from a to b.

Proof.

Let $\pi_0 = s_0 s_1 \cdots s_\delta$, and $\pi = t_0 t_1 \cdots t_\delta$, where δ is the distance between $a = s_0 = t_0$ and $b = s_\delta = t_\delta$. Let also $e_i = s_i s_{i+1}$. Note that since $c_b[e_i] = m_b[s_{i+1}]$, we have that

$$\begin{aligned}\mathbf{P}(\pi = \pi_0) &= \mathbf{P}(t_1 = s_1, \ldots . t_\delta = s_\delta) \\ &= \mathbf{P}(t_1 = s_1)\,\mathbf{P}(t_2 = s_2 \mid t_1 = s_1) \cdots \mathbf{P}(t_\delta = s_\delta \mid t_{\delta-1} = s_{\delta-1}) \\ &= \frac{c_b[e_0]}{m_b[a]} \frac{c_b[e_1]}{m_b[s_1]} \cdots \frac{c_b[e_{\delta-2}]}{m_b[s_{\delta-2}]} \frac{c_b[e_{\delta-1}]}{m_b[s_{\delta-1}]} \\ &= \frac{1}{m_b[a]},\end{aligned}$$

since $c_b[e_{\delta-1}] = 1$. ■

5.3 Complexity analysis

To calculate time and space complexity for Uniform Randomized, observe that the preprocessing algorithm actually calculates the counters c_b and m_b for all sources to the same destination b, not only from a to b.

If $n = |V|$ and $m = |E|$ are the number of vertices and edges respectively of graph $G(V, E)$, it is known (see for example [6]) that Breadth First Search on G takes $O(n+m)$ time and $O(m)$ space. The result of BFS is an orientation on up to $O(m)$ edges of G.

Since broadcasting is distributed, it takes up to the length of the longest path from b steps to set c_b and m_b on every vertex and edge of the graph. They take constant space, thus broadcasting takes $O(n + m)$ time and $O(n + m)$ space. Hence the total preprocessing from any source to destination b takes $O(n + m)$ time and space (constant space at each node and edge).

Algorithm 10 is quite fast once preprocessing has been completed. Assuming the worst-case scenario, at each intermediate vertex we can find the

value i from Step 2 in $O(\log k)$ time, where k is its number of neighbors leading to the destination. Thus if there are l hops between a and b, then the route is obtained in $O(l \log k)$ time without extra space.

Note that to preprocess for every destination, the preprocessing would need to be applied n times, taking $O(n^2 + nm)$ time and space, but this is to be expected from an all-to-all shortest path algorithm. Note also that the cost of adding a new vertex to a previously preprocessed graph G is $O(n)$, as the Breadth First Search branches have to be updated.

5.4 Routing on the lattice

Algorithm 10 can be used to find shortest routes between any pair of vertices to generate uniform random paths. The algorithm does not comply with our definition of local routing algorithms because it uses $O(n)$ space at each node and edge; however it can be applied to general networks. We can refine Algorithm 10 to work more efficiently on some types of networks. We will now show a local routing version of the algorithm for the lattice $\mathcal{L}_{n\times m}$.

Let (a, b) be a vertex of $\mathcal{L}_{n\times m}$, and denote by $N_{(a,b)}$ the number of shortest paths between $(0, 0)$ and (a, b). Note that

$$N_{(a,b)} = N_{(a-1,b)} + N_{(a,b-1)}$$

for $a \geq 1$ and $b \geq 1$. In other words, the number of paths from $(0, 0)$ to (a, b) is the sum of the paths using the horizontal edge joining $(a - 1, b)$ to (a, b) and the paths using the vertical edge which connects $(a, b - 1)$ to (a, b). We also have

$$N_{(0,0)} = N_{(0,1)} = N_{(1,0)} = 1$$

i.e. there is only one path to the neighbors of the origin. We can observe that the solution to this recurrence is

$$N_{(a,b)} = \binom{a+b}{b}$$

since $\binom{a+b}{b} = \binom{a+b-1}{b-1} + \binom{a+b-1}{b}$. Therefore we can conclude that:

Lemma 5.1 *The number of shortest paths from the origin to any vertex* (a, b) *of* $\mathcal{L}_{n\times m}$ *is* $\binom{a+b}{b}$.

Using this lemma, we can skip the preprocessing stage of Algorithm 10.

Indeed, the preprocessing stage is only needed to calculate the probability of a path using an adjacent edge of the current vertex. This probability corresponds to the proportion of paths using that edge. For example, there are $N_{(a-1,b)}$ paths going from vertex $(0,0)$ to vertex (a,b) using the horizontal edge $(0,0)(1,0)$ and $N_{(a,b-1)}$ paths also going from vertex $(0,0)$ to vertex (a,b) using the vertical edge $(0,0)(0,1)$. Since the total number of paths from $(0,0)$ to (a,b) is $N_{(a,b)}$, by Lemma 5.1, we have that the proportion of paths using the horizontal edge is

$$\frac{N_{(a-1,b)}}{N_{(a,b)}} = \frac{\binom{a+b-1}{b}}{\binom{a+b}{b}} = \frac{a}{a+b}$$

while the fraction using the vertical edge at $(0,0)$ is

$$\frac{N_{(a,b-1)}}{N_{(a,b)}} = \frac{\binom{a+b-1}{b-1}}{\binom{a+b}{b}} = \frac{b}{a+b}$$

This simplifies the calculation of probabilities for the edges to follow. In fact, at any intermediate vertex (s,t) in the path to (a,b), if we let $i = a - s$ and $j = b - t$, the probabilities of choosing the horizontal and vertical edges at

(s,t) (those leading to a vertex closer to the destination) are $\frac{i}{i+j}$ and $\frac{j}{i+j}$ respectively. Thus not only is no preprocessing necessary for routing on the lattice, but the calculation of probabilities at each vertex is also quite simple.

We now present the routing algorithm for $\mathcal{L}_{n\times m}$ which finds a uniform random route between the origin and any other vertex (a,b) in the lattice. A route between any other two vertices then follows directly.

Algorithm 11. Uniform Randomized Routing ((a,b),(0,0))
Builds π, a uniformly chosen path from (a,b) to $(0,0)$.

1. $i \leftarrow a$; $j \leftarrow b$; $\pi \leftarrow \{(a,b)\}$
2. $p \leftarrow \frac{j}{i+j}$, and let x be a Bernoulli(p) random variable
3. $i \leftarrow i-1+x$, $j \leftarrow j-x$
   Add (i,j) to π
4. Go to 2, until $i=j=0$

•

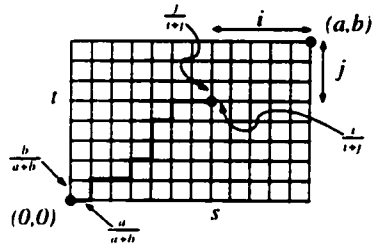


Figure 5.4: Uniform routing on the lattice

This algorithm finishes in exactly $a+b$ steps. Step 3 calculates the probability of choosing the vertical edge. Note that by Algorithm 11 we can write $\pi = v_0 v_1 \cdots v_{a+b}$, where $v_0 = (0,0)$, $v_{a+b} = (a,b)$ and

$$v_r = \left(a - r + \sum_{i=1}^{r} x_i \ , \ b - \sum_{i=1}^{r} x_i \right),$$

where $x_1 \stackrel{\mathcal{L}}{=} \text{Bernoulli}\left(\frac{b}{a+b}\right)$, and

$$x_r \stackrel{\mathcal{L}}{=} \text{Bernoulli}\left(\frac{b - \sum_{i=1}^{r-1} x_i}{a+b-(r-1)} \right)$$

for $r = 2, \ldots, a+b$. Using the above, or as a corollary of Theorem 5.1, we can prove the following.

Theorem 5.2 *Let π be the random route that* Routing(a,b) generates, and let π_0 be any shortest path route between $(0,0)$ and (a,b). Then,

$$\mathbf{P}(\pi = \pi_0) = \frac{1}{\binom{a+b}{b}} \tag{5.1}$$

5.5 Generalization: d-dimensional integer lattices

J. Urrutia and I. Stojmenovic asked the following question: is there an equivalent of the existing geometric routing algorithms in d-dimensional space? In this section we present effortless generalizations of compass routing and our uniform routing algorithm on the d-dimensional integer lattice. Of course, a flattened d-dimensional graph can always use Algorithm 10 for routing.

If $n = (n_1, n_2, \ldots, n_d)$, let $\mathcal{M}_n$ be a *d-dimensional integer lattice*. That is, for any $d \geq 2$, $i \in \{1, 2, \ldots, d\}$ and n_i a non-negative integer, the vertices of $\mathcal{M}_n$ are the d-tuples $a = (a_1, a_2, \ldots, a_d)$, where $a_i \in \{0, 1, \ldots, n_i - 1\}$. The

pair ab is an edge of $\mathcal{M}_n$ if $||b - a|| = 1$ (vertices a and b are at euclidean distance 1).

We can observe that some of the algorithms for 2-dimensional integer lattices also apply to d dimensions. Indeed, Compass Routing, Randomized Greedy, 50–50 and Uniform Randomized Routing Algorithms have natural generalizations.

5.5.1 d-dimensional Compass Routing

Compass routing, for example, always reaches the destination. Here is the algorithm:

Algorithm 12. d-Compass Routing(a,b)
Calculate π, a shortest path from vertex a to vertex b in the d-dimensional integer lattice using Compass Routing.

1. $\pi \leftarrow \{a\}$, $s \leftarrow a$

2. Let s' be the neighbor of s with the smallest planar angle $\measuredangle s'sb$ (or $\measuredangle bss'$, whichever is smaller), the angle in the 2-dimensional plane containing all three vertices. Break ties randomly.

3. $s \leftarrow s'$; add s to π

4. Repeat from Step 2 while $s \neq b$

•

5.5.2 d-dimensional Uniform Randomized

The uniform randomized routing algorithm for lattices presented above naturally generalizes to d-dimensional integer lattices.

First we need to prove that the number of paths from the origin $(0, 0, \ldots, 0)$ to any vertex $(a_1, a_2, \ldots, a_d)$ is

$$p(a) = \frac{\left(\sum_{i=1}^{d} a_i\right)!}{\prod_{i=1}^{d} (a_i!)}$$

Since this number is exactly the same for $\mathcal{M}_n$ and $\mathcal{M}_{(a_1+1, a_2+1, \ldots, a_d+1)}$, we can disregard the rest of $\mathcal{M}_n$, or better, assume without loss of generality that $a_i = n_i - 1$.

By induction over $k = \sum_{i=1}^{d} a_i$, the smallest nontrivial case is for $k = 1$, that is, there exist i such that $a_i = 1$ and $a_j = 0$, for all $j \neq i$. The lattice becomes the star of unit edges around the origin and parallel to the axes. Thus there is only one path from the origin to a, and in this case $p(a) = k / \prod_{i=1}^{d} (a_i!) = 1$, the basis of the induction.

Assume the result is true for any number less than $k = \sum_{i=1}^{d} a_i$. To prove it for k, observe that the number of shortest paths from the origin reaching a is the sum:

$$\sum_{i=1}^{d} p(a_1, a_2, \ldots, a_{i-1}, a_i - 1, a_{i+1}, a_{i+2}, \ldots, a_d)$$

That is, the number of paths reaching a is the sum of the number of paths reaching each of its neighbors. But by the induction hypothesis:

$$p(a_1, a_2, \ldots, a_{i-1}, a_i - 1, a_{i+1}, a_{i+2}, \ldots, a_d)$$
$$= \frac{\left(a_i - 1 + \sum_{j \neq i} a_j\right)!}{(a_i - 1)! \prod_{j \neq i} (a_j!)}$$
$$= \frac{a_i \cdot \left(-1 + \sum_{j=1}^{d} a_j\right)!}{\prod_{j=1}^{d} (a_j!)}$$

then

$$p(a) = \sum_{i=1}^{d} \frac{a_i \left(-1 + \sum_{j=1}^{d} a_j\right)!}{\prod_{j=1}^{d} (a_j!)}$$
$$= \frac{\left(\sum_{j=1}^{d} a_j\right) \left(-1 + \sum_{j=1}^{d} a_j\right)!}{\prod_{j=1}^{d} (a_j!)}$$
$$= \frac{\left(\sum_{j=1}^{d} a_j\right)!}{\prod_{j=1}^{d} (a_j!)}$$

Thus we have proved:

Theorem 5.3 *The number of paths from the origin to the vertex* $(a_1, a_2, \ldots, a_d)$ *of a d-dimensional integer lattice is* $\frac{\left(\sum_{i=1}^{d} a_i\right)!}{\prod_{i=1}^{d} (a_i!)}$

Given the symmetry of $\mathcal{M}_n$, this number is the same as the number of paths from a to the origin.

This is a generalization of the result obtained for $d = 2$ in the integer lattice, since in that case $p(i,j) = (i+j)!/(i!j!) = \binom{i+j}{j}$.

The result simplifies the calculation of the percentage of paths using a particular edge among those available at the current position.

Suppose a is the current vertex; then this percentage can be calculated as the number of paths going through the edge divided by $p(a)$, that is, if $r_a(i)$ represents this percentage for the edge leading to a neighbor of a differing on the i-th coordinate, then:

$$\begin{aligned} r_a(i) &= \frac{p(a_1, a_2, \ldots, a_{i-1}, a_i - 1, a_{i+1}, \ldots, a_d)}{p(a)} \\ &= \frac{\frac{\left(\sum_{j=1}^{d} a_j - 1\right)!}{(a_i - 1)! \prod_{j \neq i} (a_j!)}}{\frac{\left(\sum_{j=1}^{d} a_j\right)!}{\prod_{j=1}^{d} (a_j!)}} \\ &= \frac{a_i}{\sum_{j=1}^{d} a_j} \end{aligned}$$

This result is a generalization of the case $d = 2$, since in the integer planar lattice we had $r_a(i) = \frac{i}{i+j}$, for $a = (i,j)$.

To route in $\mathcal{M}_n$ we will extend Algorithm 11 as follows:

Algorithm 13. d-Uniform Randomized Routing (a,0)

Builds π, a uniformly chosen path from $a = (a_1, a_2, \ldots, a_d)$ to $(0, 0, \ldots, 0)$.

1. $\pi \leftarrow \{a\}$; $x_0 \leftarrow 0$; let $a_0 \leftarrow 0$
2. Let $s_a = \sum_{j=1}^{d} a_j$

3. Let t be a Uniform$\left(1, \binom{s_a}{a_1}\right)$ random variable

4. Let i be the first index such that

$$t \in \left(\sum_{j=0}^{i-1} \frac{a_j}{s_a}, \sum_{j=0}^{i} \frac{a_j}{s_a} \right]$$

   let $a_i \leftarrow a_i - 1$

   and Add $a = (a_1, a_2, \ldots, a_d)$ to π

5. Go to 2. while $a \neq (0, 0, \ldots, 0)$

•

Chapter 6

Stability of edge loads

Here we will prove that for a large class of randomized routing algorithms, the load of a communication edge is a stable random variable in the sense that it converges in probability to its mean.

6.1 Introduction

Experimental tests conducted on randomized routing algorithms usually involve several runs under controlled conditions. Average behavior over all tests is then calculated, which by the *Weak Law of Large Numbers*[1] is taken as a good indicator of how a system behaves. A common problem is that of determining how many experiments would be enough to satisfy the tester or to give a certain degree of confidence for predicting the behavior in a real

[1] "Weak Law of Large Numbers: the percentage of successes in a repeated experiment differs from the probability of the experiment being successful by more than a fixed positive amount, with a probability converging to zero as the number of experiments increases".

environment.

In the course of the research done in this thesis, a series of experiments was carried out in which randomized routing algorithms were tested over the lattice $\mathcal{L}_{n\times m}$ with unlimited capacity per edge. Our objective was to compare different algorithms to see which one behaved best in the sense that no set of edges of the lattice was overused, i.e. which algorithm distributed the traffic most evenly among the edges of our networks.

An experiment was performed for each algorithm $\mathcal{A}$ with input values for M, n and m. A simulator program then generated $t_{a,b}$ paths from vertex a to vertex b according to algorithm $\mathcal{A}$, where $t_{a,b} \in \{1, 2, \ldots, M\}$.

At the end of j-th experiment the load $l_e^{(j)}$ was calculated for each edge, and then its average load $\mu(e)$ taken over all the experiments. Surprisingly, the maximum difference $|l_e^{(j)} - \mu(e)|$ did not grow by more than 2% of the range of loads, for all j, even for relatively small values of n and m.

In order to observe this stability graphically, horizontal and vertical edge loads were plotted separately for each experiment and algorithm. The appearance of all the experiments for each $\mathcal{A}$ was the same (see Figures 6.1 and A.1).

6.2 Stability

The result in this chapter explains why "not many" experiments (in fact, only one) are required to achieve an high degree of confidence. It depends heavily on a result due to Boucheron, Lugosi and Massart [4].

Consider a sequence of spaces $\{\Omega_1, \Omega_2, \ldots\}$ and a decidable property P

defined over any subsequence $\Omega = \Omega_{i_1} \times \cdots \times \Omega_{i_n}$ of it, where $i_1 < i_2 < \cdots < i_n$. The function $P : \Omega \longrightarrow \{0, 1\}$ is *hereditary*, if for any n and every sequence $x = \{x_1, \ldots, x_n\} \in \Omega$, satisfying[2] property P, we have that any subsequence $\{x_{j_1}, \ldots, x_{j_m}\}$ of x also satisfies P.

Now we can define a *configuration function* $f : \Omega \longrightarrow \{0, 1, \ldots, n\}$ as the length of one of the longest sub-sequences of $\{x_1, \ldots, x_n\}$ that satisfies property P.

Examples of configuration functions include the length of increasing subsequences and the length of convex sub-sequences (an increasing followed by a decreasing subsequence) of a sequence of real numbers.

The following *concentration* result for configuration functions is due to Boucheron, Lugosi and Massart:

Theorem 6.1 ([4]) *Let f be a configuration function defined on $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n$. Let (Ω, P) be a space with a property and $X = (X_1, \ldots, X_n)$ be a random vector over (Ω, P) with independent components. Then for any $t \geq 0$ we have*

$$\mathbf{P}\left(f(X) \geq \mathbb{E}\{f(X)\} + t\right) \leq \exp\left(-\frac{t^2}{2\mathbb{E}\{f(X)\} + 2t/3}\right),$$

and

$$\mathbf{P}\left(f(X) \leq \mathbb{E}\{f(X)\} - t\right) \leq \exp\left(-\frac{t^2}{2\mathbb{E}\{f(X)\}}\right)$$

In other words, this result is a version of law of large numbers for configuration function $f(X)$; that is, the probability that $f(X)$ differs from its expectation by more than t tends to zero. This result gives us the tools to model load distribution by a configuration function.

[2] x satisfies P iff $P(x) = 1$.

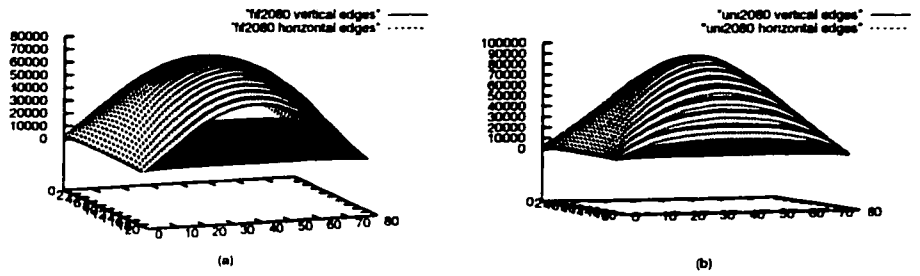


Figure 6.1: Loads on horizontal and vertical edges on a 20×80 mesh for (a) 50–50 and (b) uniform randomized routing algorithms

---

Consider a randomized routing algorithm $\mathcal{A}$ that is implemented over a graph $G(V, E)$. Let $e \in E$ be an edge of the graph and l_e its load. Let Ω be the set of paths, one per pair of vertices of G and $P_e : \Omega \longrightarrow \{0, 1\}$; $P_e(x) = 1$ iff path x use e, $P_e(x) = 0$ otherwise. Since l_e can be seen as the number of elements of Ω (paths) using edge e, l_e can be seen as a configuration function for property P_e as defined above.

Therefore we have that the inequalities in Theorem 6.1 hold for l_e; that is we have:

$$\mathbf{P}\left(l_e \geq \mathbb{E}\{l_e\} + t\right) \leq \exp\left(-\frac{t^2}{2\mathbb{E}\{l_e\} + 2t/3}\right),$$

and

$$\mathbf{P}\left(l_e \leq \mathbb{E}\{l_e\} - t\right) \leq \exp\left(-\frac{t^2}{2\mathbb{E}\{l_e\}}\right)$$

We can also prove the following theorem.

Theorem 6.2 *Let $\mathcal{A}$ be any randomized routing algorithm generating simple*

paths over the graph $G = (V, E)$. *Consider* $e \in E$ *and its load* l_e. *Then,*

$$\frac{l_e}{\mathbb{E}\{l_e\}} \xrightarrow{P} 1,$$

as $n \to \infty$ *provided* $\mathbb{E}\{l_e\} \to \infty$ *and for any* $t \geq 0$,

$$\mathbf{P}\left(\left|\frac{l_e - \mathbb{E}\{l_e\}}{\sqrt{\mathbb{E}\{l_e\}}}\right| \geq t\right) \leq 2\exp\left(-\frac{t^2}{2+o(1)}\right).$$

Proof. By Theorem 6.1.

$$\mathbf{P}\left(\left|\frac{l_e}{\mathbb{E}\{l_e\}} - 1\right| > t\right) \leq \exp\left(-3t^2\mathbb{E}\{l_e\}/8\right) + \exp\left(-t^2\mathbb{E}\{l_e\}/2\right),$$

which tends to 0 as $\mathbb{E}\{l_e\}$ goes to infinity for any $t > 0$. ■

This result tells us that l_e approaches $\mathbb{E}\{l_e\}$ very quickly as the graph grows. Since $\mathbb{E}\{l_e\}$ is fixed, this result implies that load is a stable measure. Thus if the graph is large enough, there is no need for further experiments. Load curves will be similar.

Chapter 7

Conclusions

While research in routing algorithms is a classical area of study in Computer Science, the growing demands due to the constantly increasing requirements on routing protocols for communication networks, such as the Internet, continue to push for the development of new routing schemes to satisfy the ever-changing nature of computer networks. Algorithms such as Dijkstra's and Bellman-Ford clones, which are "centralized", are not adequate for many of the requirements of modern networks.

Geometric information on nodes is a factor that is only beginning to be considered relevant for routing schemes, but as we have shown, it has a great deal of potential for producing better routing algorithms.

The uniform randomized routing algorithm shown in Chapter 5 tries to distribute traffic better that its competitors. However it remains an open question whether this is the best that can be done. It also remains open to determine the complexity of minimizing the maximum load for a graph.

Appendix A

Traffic comparison

In this appendix we make a graphical comparison of the different schemes shown in this work as a reference to the reader. The comparison is carried out by simulating traffic between every pair of nodes in both a square and a rectangular integer lattice. We also simulate the traffic between two fixed nodes in a rectangular lattice.

A.1 Routing between every pair of nodes

Consider the integer lattices $\mathcal{L}_{50\times 50}$ and $\mathcal{L}_{20\times 80}$. We run four randomized algorithms over each graph, generating exactly one path between each ordered pair of vertices a and b. Then we generate the 3D graph of the load for each edge (see Figure A.1 and A.2, where overlapped curves indicate vertical or horizontal edge loads), called *overall load*.

Observe that the volume under all curves remains constant since it is proportional to the sum of all edge loads (which is $\frac{2}{3}n^5 - \frac{2}{3}n^3$ in the case

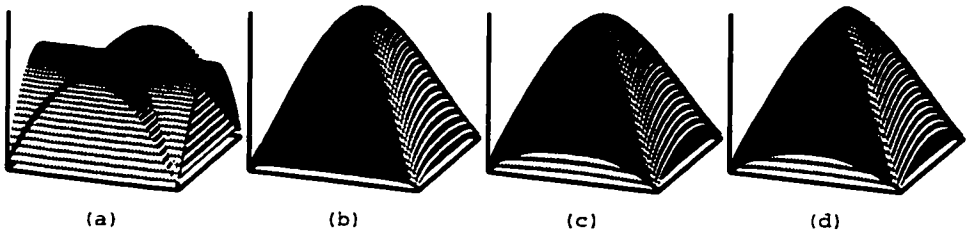


Figure A.1: Overall load for (a) Randomized Greedy (b) Uniform Randomized (c) 50–50 and (d) Compass Routing algorithms on the square lattice

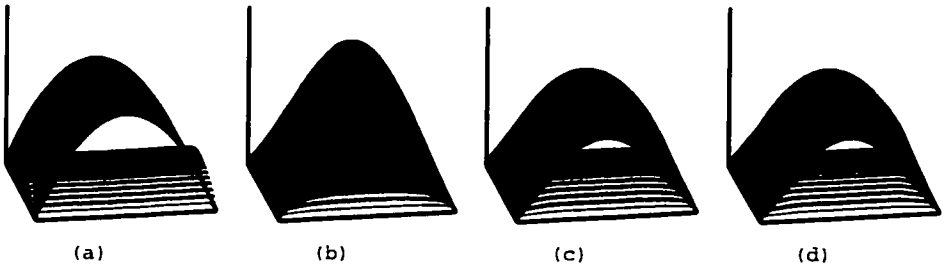


Figure A.2: Overall load for (a) Randomized Greedy (b) Uniform Randomized (c) 50–50 and (d) Compass Routing algorithms on the rectangular lattice

---

of the square lattice: see page 42). As we saw in Chapter 6, curves for different runs of each algorithm will be very similar, in the sense explained in that chapter, therefore those curves can be taken as representative of the algorithms considered.

Observe that Uniform Randomized produces a high peak and that border

edges are used fewer times than in other algorithms. The heaviest loaded edges are in the center. A network allowing heavier loads in the center edges would benefit from this information. It would be interesting to find schemes that could shift, lower or go around the peak. Even though the performance of this algorithm does not look impressive, tests in the next section will show the real potential of Randomized Routing.

A.2 Routing between one pair of nodes

Consider the integer lattice $\mathcal{L}_{100\times10}$ and run a randomized algorithm $\mathcal{A}$ to find routes from $a = (0,0)$ to $b = (99,9)$.

In Figure A.3 we show the results of a simulation of one hundred of these paths when $\mathcal{A}$ is "50–50" (top left), "Compass Routing" (top right), "Randomized Greedy" (bottom left) and "Uniform Randomized" (bottom right).

A.2.1 50–50 routing

Note that for routes generated by the 50–50 routing algorithm it is highly probable that as many horizontal as vertical edges will be chosen. Then the first edges of every route will approach the diagonal at 45° from the origin, that is, the one with endpoints $(0,0)$ and $(9,9)$. After reaching the top of the lattice, there is no other choice for a shortest path than horizontal edges.

In this way edge or node failures on elements in the upper horizontal part of the lattice are weaknesses of this scheme. It would not be easy to overcome this weakness without compromising factors such as path shortness

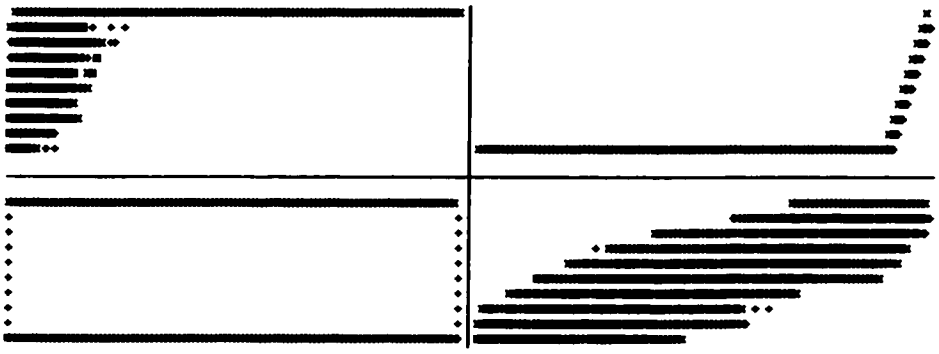


Figure A.3: 100 paths from lower-left to upper-right corners of $\mathcal{L}_{100\times10}$. Sections of this figure were generated with "50–50", "Compass Routing", "Randomized Greedy" and "Uniform Randomized" routing algorithms respectively.

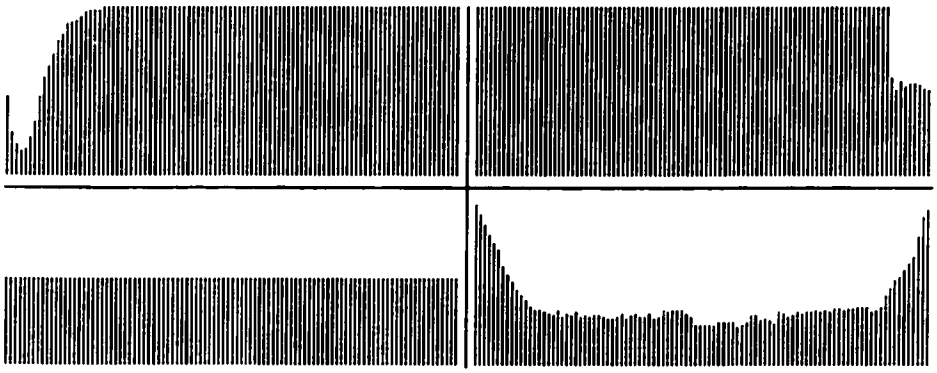


Figure A.4: Maximum height diagrams for the corresponding algorithms in previous figure

---

or guarantee of delivery.

A.2.2 Compass Routing

The hundred routes using Compass Routing algorithm from $a = (0,0)$ to $b = (99,9)$ traverse first the path $(0,0)(1,0)\cdots(89,0)$ and then edges touching the diagonal joining the vertices $(90,0)$ and $(99,9)$ (see Figure 2.7). The right side of the corresponding section of Figure A.4 shows the result of tie-breaking for the last 9 columns of the mesh.

It is evident that the weakness of this algorithm, in the sense of edge or node failures, is the path generated by the compass. However the ability to correct this problem can be seen as embedded into the algorithm, since the compass is applied at every step and it can provide an alternative route, even if a slightly longer path results.

A.2.3 Randomized Greedy

The routes generated by this algorithm are split equally among the following:

$$(0,0)(0,1)\cdots(0,9)(1,9)\cdots(99,9)$$

and

$$(0,0)(1,0)\cdots(99,0)(99,1)\cdots(99,9)$$

The curve in Figure A.4 shows that traffic is shared equally between the two paths. It is evident that all other edges of the lattice are not used at all.

This is perhaps the least robust of the algorithms shown here, as there are only two possible paths to choose from. How would this scheme or a modification behave in the presence of edge or node failures? The randomness introduces an alternative path but only at the beginning, thus the first and

last horizontal and vertical paths are the weakness of this scheme. A possible modification would be to follow its designated (horizontal-then-vertical or vertical-then-horizontal) path "as many times as it can" and take the alternative edge only in case of failures. This however, would increase the complexity of this quite simple scheme.

A.2.4 Uniform Randomized

The first noticeable feature in the bottom right of Figure A.3 is the broad range of routes used by the "Uniform Randomized" routing algorithm compared to the other algorithms, which have fewer available routes. Second, the algorithm mostly uses routes at the center of the lattice (near the diagonal). The explanation for this behavior is that there are more routes in that region than at the borders, so the probability of going through the center is larger than through the borders. However every possible path has equal probability to be chosen. Also notice that the "base" of the figure is somewhat larger that its "lid". The reason for this is that most of the routes start horizontally, because of the dimensions of the lattice. It is important to note that this shape will get thicker as the lattice gets thinner and vice versa. Finally, in Figure A.4 we can see that traffic decreases to less than 50% for edges outside the squares $(0,0)$–$(9,9)$ and $(90,0)$–$(99,9)$. This shows that the performance of this algorithm is better on long strips. The empty space above the curve in this figure is the largest among the four algorithms presented, which indicates that few edges receive a high load.

The robustness of this algorithm, measured as the number of edges that could be eliminated without interrupting the communication between a and

b, is the highest of the algorithms seen here. The algorithm also handles edge failures nicely as it finds them in a way similar to Compass Routing. However it does not compromise any aspect of the routing (path shortness, guarantee of delivery) if at least one of the edges leading to a shortest path to the destination is available.

Index

Bibliography

[1] F. Baker. *Request for comments*, rfc1812, http://www.ietf.org/rfc/rfc1812.txt. June 1995.

[2] Prosenjit Bose. Pat Morin, Andrej Brodnik, Svante Carlsson, Erik D. Demaine, Rudolf Fleischer, J. Ian Munro, and Alejandro Lopez-Ortiz. *Online routing in convex subdivisions*, International Symposium on Algorithms and Computation, 2000, pp. 47–59.

[3] Prosenjit Bose. Pat Morin, Ivan Stojmenovic, and Jorge Urrutia, *Routing with guaranteed delivery in ad hoc wireless networks*, Wireless Networks **7** (2001), no. 6. 609–616.

[4] Stephane Boucheron, Gabor Lugosi, and Pascal Massart, *A sharp concentration inequality with applications*, Random Structures and Algorithms **16** (2000), no. 3. 277–292.

[5] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva, *A performance comparison of multi-hop wireless ad hoc network routing protocols*, Mobile Computing and Networking, 1998, pp. 85–97.

[6] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to algorithms*, M.I.T. Press, Cambridge, Massachusetts, U.S.A., 1990.

[7] S. Datta, I. Stojmenovic, and J. Wu, *Internal node and shortcut based routing with guaranteed delivery in wireless networks*, Cluster Computing **5** (2002), no. 2, 169–178.

[8] Greg N. Frederickson and Ravi Janardan, *Designing networks with compact routing tables*, Algorithmica **3** (1988), 171–190.

[9] Cyril Gavoille, *A survey on interval routing*, Theoretical Computer Science **245** (2000), no. 2, 217–253.

[10] Cyril Gavoille and David Peleg, *The compactness of interval routing*, SIAM Journal on Discrete Mathematics **12** (1999), no. 4, 459–473.

[11] C. Hedrick, *Request for comments*, rfc1058, http://www.ietf.org/rfc/rfc1058.txt, June 1988.

[12] T. Imielinski and J. Navas, *Request for comments*, rfc2009, http://www.ietf.org/rfc/rfc2009.txt, November 1996.

[13] David B Johnson and David A Maltz, *Dynamic source routing in ad hoc wireless networks*, Mobile Computing (Imielinski and Korth, eds.), vol. 353, Kluwer Academic Publishers, 1996.

[14] Nabil Kahale and Tom Leighton, *Greedy dynamic routing on arrays*, Journal of Algorithms **29** (1998), no. 2, 390–410.

[15] Young-Bae Ko and Nitin H. Vaidya, *Location-aided routing (LAR) in mobile ad hoc networks*, Mobile Computing and Networking, 1998, pp. 66–75.

[16] Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia, *Compass routing on geometric networks*, Proc. 11th Canadian Conference on Computational Geometry (Vancouver), August 1999, pp. 51–54.

[17] Pat Morin, *Online routing in geometric graphs*, Ph.D. thesis, School of Computer Science, Carleton University, January 2001.

[18] C. H. Papadimitriou and M. Yannakakis, *Shortest paths without a map*, Theoretical Computer Science **84** (1991), no. 1, 127–150.

[19] Vincent D. Park and M. Scott Corson, *A highly adaptive distributed routing algorithm for mobile wireless networks*, INFOCOM (3), 1997, pp. 1405–1413.

[20] Charles Perkins and Pravin Bhagwat, *Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers*, ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications, 1994, pp. 234–244.

[21] E. Royer and C. Toh, *A review of current routing protocols for ad-hoc mobile wireless networks*, IEEE Personal Communications, 1999.

[22] N. Santoro and R. Khatib, *Labelling and implicit routing in networks*, The Computer Journal **28** (1985), 5–8.

[23] Mahtab Seddigh, Ivan Stojmenovic, and Jie Wu, *Location and internal nodes based routing algorithms in wireless networks, to appear.*

[24] Harvinder Singh. *Compass routing on geometric graphs*, Master's thesis, School of Information Technology and Engineering, University of Ottawa, 1998.

[25] I. Stojmenovic. *Voronoi diagram and convex hull based geocasting and routing in wireless networks*, Tech. Report TR-99-11, University of Ottawa, School of Information Technology and Engineering, 1999.

[26] Ivan Stojmenovic and Xu Lin., *Power-aware localized routing in wireless networks*, IEEE Transactions on Parallel and Distributed Systems **12** (2001), no. 11. 1122–1133.

[27] J. van Leeuwen and R. Tan. *Interval routing*, The Computer Journal **30** (1987), 298–307.

[28] Min-You Wu, *On runtime parallel scheduling for processor load balancing*, IEEE Transactions on Parallel and Distributed Systems **8** (1997). no. 2, 173–186.