Routing and Search on Large Scale Networks

Routing and Search on Large Scale Networks

Dominique Tschopp

EPFL - Ecole Polytechnique Fédérale de Lausanne

Thesis No. 4589 (January 2010)

Thesis presented to the faculty of computer and communication sciences for obtaining the degree of Docteur ès Sciences

Accepted by the jury:

M. Grossglauser and **S. Diggavi**
Thesis directors

J.-Y. Le Boudec
Expert

M. Mitzenmacher
Expert

S. Shakkottai
Expert

E. Telatar
President of the jury

Ecole Polytechnique Fédérale de Lausanne, 2010

to my wife, Joyce, and my family...

Abstract

In this thesis, we address two seemingly unrelated problems, namely routing in large wireless ad hoc networks and comparison based search in image databases. However, the underlying problem is in essence similar and we can use the same strategy to attack those two problems. In both cases, the intrinsic complexity of the problem is in some sense low, and we can exploit this fact to design efficient algorithms.

A wireless ad hoc network is a communication network consisting of wireless devices such as for instance laptops or cell phones. The network does not have any fixed infrastructure, and hence nodes which cannot communicate directly over the wireless medium must use intermediate nodes as relays. This immediately raises the question of how to select the relay nodes. Ideally, one would like to find a path from the source to the destination which is as short as possible. The length of the found path, also called *route*, typically depends on how much signaling traffic is generated in order to establish the route. This is the fundamental trade-off that we will investigate in this thesis. As mentioned above, we try and exploit the fact that the communication network is intrinsically low-dimensional, or in other words has low complexity. We show that this is indeed the case for a large class of models and that we can design efficient algorithms for routing that use this property. Low dimensionality implies that we can well embed the network in a low-dimensional space, or build simple hierarchical decompositions of the network. We use both those techniques to design routing algorithms.

Comparison based search in image databases is a new problem that can be defined as follows. Given a large databases of images, can a human user retrieve an image which he has in mind, or at least an image similar to that image, without going sequentially through all images? More precisely, we ask whether we can search a database of images only by making comparisons between images. As a case in point, we ask whether we can find a query image q only by asking questions of the type "does image q look more like image A or image B"? The analogous to signaling traffic for wireless networks would here be the questions we can ask human users in a learning phase anterior to the search. In other words, we would like to ask as few questions as possible to pre-process and preprare the database, while guaranteeing a certain quality of the results obtained in the search phase. As the underlying image space is not necessarily metric, this raises new questions on how to search spaces for which only rank information can be obtained. The rank of A with respect to B is k, if A is B's k^{th} nearest neighbor. In this setup, low-dimensionality is analogous to the homogeneity of the image space. As we will see, the homogeneity can be captured by properties of the rank relationships. In turn, homogeneous spaces can be well decomposed hierarchically using comparisons. Further, it allows us to design good hash functions.

To design efficient algorithms for these two problems, we can apply the same techniques *mutatis mutandis*. In both cases, we relied on the intuition that the problem has a low intrinsic complexity, and that we can exploit this fact. Our results come in the form of simulation results and asymptotic bounds.

Keywords: *Wireless Networks, Search, Dimensionality Reduction, Databases, Routing, Algorithms, Asymptotic Bounds, Similarity*

Résumé

Dans cette thèse, nous abordons deux problèmes qui semblent à première vue ne pas avoir de liens entre eux. Ces deux problèmes sont le routage dans les réseaux sans fil *ad hoc*, et la recherche basée sur des comparaisons dans des bases de données d'images. Pourtant, le problème sous-jacent est essentiellement similaire et nous pouvons utiliser la même stratégie pour attaquer ces deux problèmes. Dans les deux cas, la complexité intrinsèque du problème est dans un certain sens basse. Nous pouvons exploiter cette propriété pour concevoir des algorithmes performants.

Un réseau sans fil ad hoc est un réseau de communication composé d'appareils sans fil, tels des ordinateurs portables ou des téléphones mobiles. Le réseau n'a pas d'infrastructures fixes, et par conséquent les noeuds qui ne peuvent pas communiquer directement doivent pouvoir utiliser d'autres noeuds comme relais pour transmettre leurs paquets. Un élément important dans la conception et le déploiement de tels réseaux est bien sûr la méthode de séléction de ces relais. Idéalement, nous aimerions obtenir une séquence de relais, ou *route*, de la source à la destination qui soit aussi courte que possible. La longueur de la route dépend typiquement de la quantité de trafique additionelle que nous sommes prêts à générer. C'est ce compromis fondamental entre trafique et longueur des routes auquel nous nous intéressons dans cette thèse. Nous exploitons le fait que le réseau de communication est intrinsèquement de basse dimensions, ou en d'autres termes, de basse complexité. Nous montrons que c'est bien le cas pour une grande class de modèles et que nous pouvons concevoir des algorithmes performants qui exploitent cette propriété. La basse dimensionalité implique que nous pouvons incorporer le réseau dans un espace de basse dimension, ou construire une décomposition hiérarchique simple. Nous utilisons ces deux techniques pour déveloper des algorithmes de routage.

La recherche basée sur des comparaisons dans des bases de données d'images est un nouveau problème qui peut être défini comme suit. Est-ce qu'un utilisateur humain peut trouver, dans une grande base de données, une image qu'il a en tête, ou au moins trouver une image similaire à cette image, sans parcourir toutes les images ? Plus précisément, nous posons la question de savoir s'il est possible de chercher une image dans une base de données, uniquement en faisant des comparaisons. C'est-à-dire, en posant des questions du type "est-ce que l'image recherchée q ressemble plus à l'image A ou à l'image B ?". L'équivalent du trafique dans les réseaux sans fil sont ici les questions posées durant la phase d'apprentissage aux utilisateurs humains. Nous aimerions poser aussi peu de questions que possible pour préparer les données avant de débuter les recherches, tout en garantissant une certaine qualité dans le résultat des recherches. Etant donné que l'espace sous-jacent contenant les images n'est pas nécessairement métrique, ce problème nous amène à réfléchir sur la recherche dans des espaces où seul des informations sur les rangs respectifs peuvent être obtenues. Le rang de A par rapport à B est k, si A est le $k^{ème}$ élément le plus proche de B. Dans ce contexte, la basse dimensionalité peut être associée à l'homogénéité de l'espace contenant les images. Nous verrons que cette homogénéité peut être décrite par des propriétés définies sur les rangs. Les espaces homogènes peuvent être décomposés en hiérarchie, et permettent de développer des fonctions de hashage performantes.

Pour concevoir des algorithmes performants pour ces deux problèmes, nous pouvons appliquer les mêmes techniques *mutatis mutandis.* Dans les deux cas, nous nous sommes basés sur l'intuition que le problème possède une complexité intrinsèque basse, et que nous pouvions nous servir de cette particularité du problème. Nos résultats se présentent sous la forme de résultats de simulations et de bornes asymptotiques.

Mots-clés: *Réseaux sans fil, Recherche, Réduction de Dimensionalité, Bases de Données, Routage, Algorithmes, Bornes Asymptotiques, Similarité*

Acktionary

First, I would like to heartfully thank my supervisors, Prof. Suhas Diggavi and Prof. Matthias Grossglauser, for welcoming me in their groups and helping me with my research in the past 5 years. Working with them has taught me to think about complex problems in a new, more structured and creative way. Second, I would like to thank my wife Joyce, my parents Erna and Juerg, and my sister Muriel, for their support during this long process. In addition to my supervisors, I would also like to thank the other members of my thesis jury, Prof. Michael Mitzenmacher, Prof. Sanjay Shakkottai, and Prof. Jean-Yves Le Boudec for kindly accepting to review my thesis, and Prof. Emre Telatar for accepting to be the president of this jury. I thank Docomo Eurolabs for funding the first part of my thesis work, and the MICS NCCR for funding the second part. Finally, I would like to thank my colleagues and friends at EPFL, with whom it was a great pleasure interacting, both at work and outside. In particular, I would like to thank the members, present and past, of our lunch group (also known as the *LCA lunchers*): Irina, Mathilde, Daniel, George, Gianluca, Hung, Manuel, Nikodin, Patrick, Ramin and Ruben.

Contents

Abstract i

Résumé iii

Acktionary v

Contents vii

1 Introduction 1
- 1.1 Routing in Mobile Wireless Ad hoc Networks 1
- 1.2 Comparison-based Search in Image Databases 4
- 1.3 A Needle in a Flat Haystack 6

2 Virtual Coordinates for Routing in Dynamic Ad hoc Networks 9
- 2.1 Relationship to Published Works 12
- 2.2 Low-Dimensional Embeddings 13
  - 2.2.1 Stable Dynamic Embedding of Connectivity Graph . . . 15
  - 2.2.2 Observations on Wireless Connectivity Graph 16
- 2.3 Embedding Algorithm 18
  - 2.3.1 Embedding Heuristic 18
  - 2.3.2 Dealing with Dynamic Graphs 21
  - 2.3.3 Formal Description of PB Algorithm 21
- 2.4 Routing Algorithm 22
- 2.5 Simulation Results 24
  - 2.5.1 Experiment Design 25
  - 2.5.2 Performance Metrics 25
  - 2.5.3 Static Networks 26
  - 2.5.4 Mobile Networks 29
- 2.6 Concluding Remarks 32

3 Hierarchical Routing in Dynamic Ad hoc Networks 33
- 3.1 Relationship to Published Works 35
- 3.2 Models and Definitions 37

- 3.2.1 Geometric random graph 38
- 3.2.2 SINR full connectivity 39
- 3.2.3 Uniform speed-limited (USL) mobility 41
- 3.2.4 Assumptions 43
- 3.3 Network Properties 43
  - 3.3.1 Inhomogeneous Topologies 47
  - 3.3.2 Sequences of Communication Graphs 54
- 3.4 Routing Algorithm 56
  - 3.4.1 Beaconing Algorithm 58
  - 3.4.2 Forwarding Algorithm 61
  - 3.4.3 Load-balancing 63
- 3.5 Performance Analysis 65
  - 3.5.1 Control Traffic 66
  - 3.5.2 Route Stretch 67
- 3.6 Implementation Issues 68
- 3.7 Concluding Remarks 70

4 Comparison-based Nearest Neighbor Search 71
- 4.1 Relationship to Published Works 73
- 4.2 Definitions and Problem Statement 75
- 4.3 Contributions 77
- 4.4 Searching with Known Disorder Constant 78
  - 4.4.1 Hierarchical Data Structure For Nearest-Neighbor Search 79
  - 4.4.2 Lower Bound 83
- 4.5 Searching with Unknown Characterization 87
  - 4.5.1 Consequences of knowing the disorder constant 88
  - 4.5.2 Binary Tree Decomposition 89
  - 4.5.3 Rank-Sensitive Hashing 92
  - 4.5.4 Rank-Sensitive Hashing in high-dimensional Euclidean Spaces 94
- 4.6 Concluding Remarks 97

5 A Platform for Similarity Search 99
- 5.1 Relationship to Published Works 100
- 5.2 Image Processing 102
  - 5.2.1 Eigenfaces 102
- 5.3 System 104
- 5.4 Experimental Results 106
  - 5.4.1 Web Platform 108
- 5.5 Concluding Remarks 109

6 Conclusions and Future Work 111

Appendix 113
- A-1 Unit Disc Graphs 113

A-2 Random Geometric Graphs with subthreshold Communication Radius 114
A-3 Locality-Sensitive Hashing 117

Bibliography **119**

Curriculum Vitae **127**

Introduction

1

In this thesis we address two seemingly unrelated problems, namely routing in large mobile wireless ad hoc networks, and comparison-based search in large databases of pictures. However, as we will see, similar techniques can be used *mutatis mutandis* to work on those two problems. In particular, in both cases we can try and exploit some properties of the database or the network that make the intrinsic complexity of those structures low. For both wireless networks and databases of pictures, it seems intuitive that there must exist efficient search algorithms. Obviously, one of the main difficulties is to find the right characterization of the problem, and then to design algorithms of which the performance depends on this characterization. The low complexity typically leads to a hierarchical decomposition or a low-dimensional representation, and roughly logarithmic search times. We design algorithms that are very efficient when this intrinsic complexity is low. In Sections 1.1 and 1.2 of this introductory chapter, we will first define and motivate those two problems in a somewhat informal way. More precise and formal definitions, as well as results, are then provided in subsequent chapters.

1.1 Routing in Mobile Wireless Ad hoc Networks

A mobile wireless ad hoc network is a network of nodes such as laptops, PDA's, or cell phones, that cannot rely on any fixed infrastructure to communicate. This model is different from the classical model used in existing WiFi or cellular infrastructures, where nodes communicate via a base station. This ad hoc setup implies that every node must have the capability of acting simultaneously as a client, a server and a relay. The capacity to act as a relay is crucial, as the source and the destination of a message might be out of radio range and could consequently not necessarily communicate directly. An illustration of an ad hoc network is shown in Figure 1.1. As mentioned above, ad hoc networks do not depend on any fixed infrastructure. Clearly, there are many scenarios where this is potentially an advantage. One could think of disaster relief situations, where existing base stations would have been destroyed, and rescue workers would communicate in an ad hoc manner. Alternatively, one could think of vehicular networks, in regions where constructing base stations to ensure full coverage would be far to expensive. Another scenario which is often cited as example are sensor networks. Such networks are composed of devices with limited communication capabilities and obviously sensors. They are typically deployed in regions which are hard to access and where building a fixed infrastructure would be too costly or simply too difficult because of the terrain. Finally, the scenario that is maybe the most interesting one is the use of ad hoc communications to extend the range of base stations. This might be particularly interesting in countries where buying land to setup a fixed antenna is very costly and legally difficult.

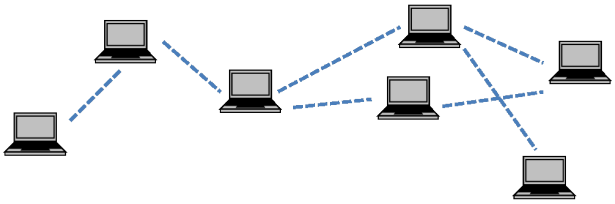


Figure 1.1: An illustration of a wireless ad hoc network composed of seven laptops. Due to a limited radio range and to interferences, the laptops on the right hand side and on the left hand side cannot communicate directly with each other. Rather, messages must be relayed by intermediate nodes.

Due notably to the mobility of the nodes and the random nature of the wireless channel, designing protocols for such networks is very challenging. One of the most difficult problems to solve for ad hoc networks is routing[1]. Indeed, in contrast to networks with fixed infrastructure, communications cannot be routed through a central base station known to all nodes. Further, one cannot even rely on a central repository containing the addresses or locations of all nodes. As nodes are inherently mobile, routes change over time and consequently fixed routing tables are not useful, as routes are often only valid for a short period of time. Recall that distant nodes cannot communicate directly with each other and intermediate nodes must act as relays. Hence, we need to design a routing scheme that allows us to maintain short paths between nodes, while at the same time not adding a large overhead in terms of signaling traffic.

---

[1] In this thesis we focus on the scenario that multihop routing is the throughput optimal strategy. It is perhaps appropriate to note that node cooperation might be important in other scenarios [ADT07, ÖLT07, ADT09]

The latter point is of great importance in wireless networks, where bandwidth is a scarce resource. The first part of this thesis will be dedicated to the design of efficient routing schemes for mobile ad hoc wireless networks.

Intuitively, it seems that wireless ad hoc networks must be low dimensional. Indeed, it seems that the embedding of the network into $\Re^2$ provided by the global positioning system (GPS) coordinates should roughly capture the topology of the network. In other words, the Euclidean distance between the positions of the nodes in the "real" world should be roughly proportional to some graph distance measured in the network (*e.g.,* we could build a graph by adding an edge of weight 1 between nodes that can communicate directly and measure distances as the shortest path distance in the graph). This intuition relies on the fact that the signal strength decays super-linearly with distance, and that consequently direct communications must be local[2]. In turn, distant nodes must communicate over multiple hops, and the more distant nodes are, the more hops separates them. This idea is clearly true if the nodes are distributed homogeneously and sufficiently densely on a convex area, and if the radio range is small with respect to the diameter of the network. In this case, a greedy forwarding strategy, where nodes repeatedly send packets to the neighbor with the position closest to the destination of the message will be successful and will lead to short routes. In case the nodes were mobile, the major difficulty would be to locate the destination. The common solution to this problem consists in implementing a distributed database in the network that is dynamically updated when the topology changes.

In this thesis, we investigate the more general case where the network does not necessarily originate from an homogeneous distribution of nodes. In particular, in any realistic scenario, there could be obstacles such as walls, or topological voids, that prevent nearby nodes from communicating directly or over a small number of relays. Hence, a greedy forwarding strategy based on GPS coordinates would fail. Recovery mechanisms exist to get out of so-called "dead-ends", but they are also costly in terms of overhead, and are to some extent dependent on a specific model for the network. Moreover, the fact that the nodes are mobile prevents us from storing static routes, and maintaining all-to-all short routes is too costly in terms of signaling traffic. We try and exploit the fact that even though the network topology might not be perfectly homogeneous, it is still likely to be low-dimensional. That is, the network might not necessarily be embeddable in a two dimensional space but potentially in a space of dimension either constant (*i.e.,* independent of the cardinality of the network), or at least only logarithmic in the cardinality of the network. Our first approach will consist in developing a distributed algorithm that embeds the network in a low dimensional space. We expect that by "pushing nodes around a bit", the new coordinates will correspond better to the topology of the network. In particular, we now expect the Euclidean distance between nodes to be a better approximation of the graph distance. This approach is described in Chapter 2. The virtual coordinates obtained in that way will considerably

---

[2] This is true in the case when path loss decay with distance is large ($\alpha \geq 3$)

increase the quality of greedy forwarding, while not increasing the signaling traffic much. The results in Chapter 2 are based on simulations of our routing protocol and comparisons with the performance of existing routing schemes. Our second approach, described in Chapter 3, consists in developing a structure that adapts to the intrinsic dimension of the network. Here, intrinsic dimension refers to the doubling dimension, a notion to be defined more formally in Chapter 3, and which can informally be defined as the maximum number of balls of radius r required to cover a ball of radius $2r$. The results presented in Chapter 3 are in some sense a theoretical extension of the results in Chapter 2. Indeed, in this chapter we try and formalize the intuition that we will develop in Chapter 2. In contrast to the simulation results in Chapter 2, the results in Chapter 3 come in the form of asymptotic bounds. We also prove that for common models for wireless networks and mobility, the communication graph has desirable properties. For instance, we try and capture the intuition that under mobility, distant nodes will not suddenly be connected directly, if they were several hops apart. In fact, one of the key contributions of this thesis is the formal analysis of networks with dynamically changing topologies. In other words, our theoretical results do not only apply to static networks, but also to networks of mobile nodes and networks with uncertain communcation channels.

1.2 Comparison-based Search in Image Databases

To explain the second problem, comparison-based search in large databases, we first give an example that motivates this question. Assume that a large organization, a company, the police or (why not) a dating service stores a large database of faces. For instance, a company could store a database containing a picture of every employee, where every picture is linked with the office number, the phone number and an email address. Assume, further, that you recently met someone who was working on a topic of great interest to you, and that you would like to find his contact details. As a distracted person, you only remember the organization he works for, but forgot his name. Can you efficiently retrieve the contact details of this person from the employee database, without an exhaustive search *i.e.,* can you design a search engine that returns the picture of this employee very quickly? Similarly, the police could store a database of dangerous criminals. A victim could then try to browse this database to find the person who attacked her, and help the police arrest him. Potentially, no digital form of the query image is available, or automatic image processing algorithms perform poorly because some images are taken in different environments (*e.g.,* light, angle, background, exposition, distance, etc.). Further, it is hard to define a meaningful similarity measure based on image processing that fits the perception of all humans, for all images. Nevertheless, it might be possible to actually use humans as classifiers, if the number of questions humans must answer is reasonably small. In this thesis, we try and characterize the properties which the image space must have if we want to perform search based only on comparisons. The practical problem exposed above raises deeper and more theoretical questions about search in non-metric spaces, which can be characterized only by relative distance information (*i.e,* by rank relationships[3]). In particular, one could ask what desirable properties of such a space are. We try and answer some of these questions in this thesis. In essence, our plan of attack for this problem is analogous to the approach taken for wireless networks. Indeed, we first try and find an appropriate characterization of the problem that makes its complexity low, and then develop algorithms that exploit this property.

More precisely, we ask the question of whether we can search such a database of pictures only by making comparisons. In other words, can we retrieve a picture similar to a query image, only by asking questions of the type "does the image you are looking for look more similar to image B, or image C" (an example is shown in Figure 1.2). As mentioned above, the intuition underlying this approach is that humans are excellent classifiers for images, and that we can use them to do some processing which cannot be automated. Comparisons based on feature vectors or other image processing techniques often partly fail if the images are not taken under the same conditions. For instance, it is difficult to compare images that are taken from different angles, with different backgrounds and with different exposition times. It is then even more difficult to compute meaningful distances or similarities based on these vectors. In fact, similarity between pictures is a notion that is difficult to define formally, even in words, and even more difficult to measure automatically based on the content of images. Nevertheless, we expect humans to be able to classify images, at least up to a certain "precision". For instance, we expect that if a user who is looking for the picture of a woman, is asked whether this woman is more similar to a woman or a man, he is very likely to click on the woman. Similarly, we expect humans to have the capacity to classify images based on age, hair

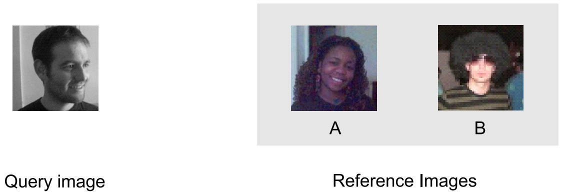


Figure 1.2: The user is asked whether the image he is looking for *i.e.,* the query image on the left hand side, looks more similar to the reference image A or B. Even though the man in reference image B is wearing a wig, a human user is very likely to click on image B. Indeed, the user can very efficiently process and classify or label the three images.

---

[3] The rank of A with respect to B is k, if A is B's k^{th} nearest neighbor.

color, eye color, size of the ears, and so on. More importantly, humans are likely to give more importance to criteria such as gender and skin color, which leads to a shared notion of similarity. It is also likely that the human brain takes into account other features that are not obvious. Hence, we intend to use humans to answer "difficult" questions in the learning phase, and to perform searches. One of the difficulties of this problem is that on one hand, there are many comparisons that can be made, and on the other hand asking humans to make comparisons is very costly in terms of time and effort. Thus, we must carefully choose the questions that we ask users, while guaranteeing a certain quality of search. Another difficulty is that the space in which the images live, and the similarities perceived by humans, might not result in a metric space. Hence, we need to think about new ways of searching, as classical approaches are not necessarily applicable.

In Chapter 4, we ask a more general theoretical question. Indeed, we ask what property a space of object (*e.g,* images, but potentially also music, videos, etc.) must have such that it can be searched efficiently by human users who can only make comparisons. Moreover, given that the space of objects has such a property, we try and develop algorithms of which the performance depends on this property. Again, as it is the case for wireless ad hoc networks, it appears that images must have an intrinsic structure that has a low complexity. For databases, we will measure this complexity in terms of two properties of the space, namely its disorder constant and rank distortion. Both properties roughly capture how homogeneous the space is. The first one, however, is a worst case criterion, and the second one an average criterion. This setup is also interesting because, in contrast to many existing formulations, we do not necessarily require the space in which the objects live to be metric. Rather, we characterize the space in terms of its rank relationships. Such relationships can be defined for any space.

In Chapter 5, we then present the architecture of a system that implements some of the ideas of Chapter 4. We also present experimental results for search based on comparisons. As we will see, a practical implementation raises a number of additional issues. We highlight a few of these issues and propose practical solutions. In particular, humans sometimes (if not often) disagree on the notion of similarity, and we need to take care of this problem. We try and reduce the number of training clicks further by combining human training with image processing.

1.3 A Needle in a Flat Haystack

The problem underlying both routing in large wireless networks and comparison based search in large databases is the same in essence. In both cases, we need to efficiently retrieve a specific element in a large collection of elements *i.e.,* in the database or in the network. Further, we need to find good "paths" in the collection towards this specific element, that is we must in some sense make the collection navigable. Indeed, in wireless networks we must establish routes from source nodes to destination nodes, and in the database, we must identify sequences of questions that allow us to retrieve specific elements. In both cases, an exhaustive search of the collection is too costly, either in terms of control traffic, or in terms of questions that we need to ask the human user. Hence, we need to develop methods to proactively process the elements in such a way that the search phase can be considerably sped up. However, preprocessing does obviously not come for free either. Surely, preprocessing also requires sending messages through the network or asking questions. On the other hand, in the preprocessing phase, the cost can be amortized over all the elements. Hence, in both cases, we need to find a trade-off between preprocessing operations and search time.

The two problems addressed in this thesis were not chosen at random. Indeed, in general it might not be possible to considerably reduce the search time by preprocessing the elements. However, for routing in wireless networks and comparison-based search in databases, it appears that the respective collections of elements have the desirable property that they are not intrinsically very complex. The most natural notion of complexity is the dimension in which the elements live. For instance, if the elements were a set of points in $\Re^d$, for large d, most problems might be difficult to solve. However, if the points lie close to a low dimensional plain in $\Re^d$, the problem might become much easier to solve. Another difficulty that we have not mentioned so far is that even though the elements of both problems live in spaces of low intrinsic complexity, we cannot "batch process" these spaces by building classical data structures such as k-d-trees or hash tables. Indeed, we are in both cases only given partial information *e.g.,* a subset of hop distances or a subset of human answers, and need to do the best possible job with it. In wireless networks, this implies that we need to develop efficient *distributed* routing algorithms, and for database search, it implies that we need to ask the right questions to human users, and be able to work only with relative distance information *i.e.,* we know that image A looks more like B than like C, but we cannot obtain meaningful numerical values for distances between images.

In Chapter 2, we make a first attempt at exploiting the intrinsically low dimensional structure of wireless networks. We present an approach for embedding connectivity graphs into a low-dimensional Euclidean space in a distributed way. We also show how the "virtual coordinates" obtained this way can be used for greedy forwarding of packets. We evaluate our techniques through simulations. In Chapter 3, we first show formally that a large class of common models for the connectivity and the mobility of wireless ad hoc networks share the property that the resulting communication graph is intrinsically low-dimensional (a notion to be defined formally in Chapter 3). Then, we present algorithms that exploit this low dimensionality and prove asymptotic bounds on the signaling traffic and the route stretch. In Chapter 4, we present our work on comparison based search for image database. In this chapter, we present a theoretical model for such spaces of images and algorithms that exploit the properties of such spaces. Again, we prove asymptotic bounds on the performance. Then, in Chapter 5, we present the architecture of a web platform that implements the ideas from Chapter 4. Finally, in Chapter 6, we present our conclusions and ideas for future work.

The three main chapters *i.e.,* Chapters 2,3 and 4, treat clearly distinguishable topics. In order to make those chapters self-contained, so that they can be read separately, each one has its own generally non-overlapping related work section. Similarly, we present a model, definitions, assumptions as well as results in each of these chapters.

2 Virtual Coordinates for Routing in Dynamic Ad hoc Networks

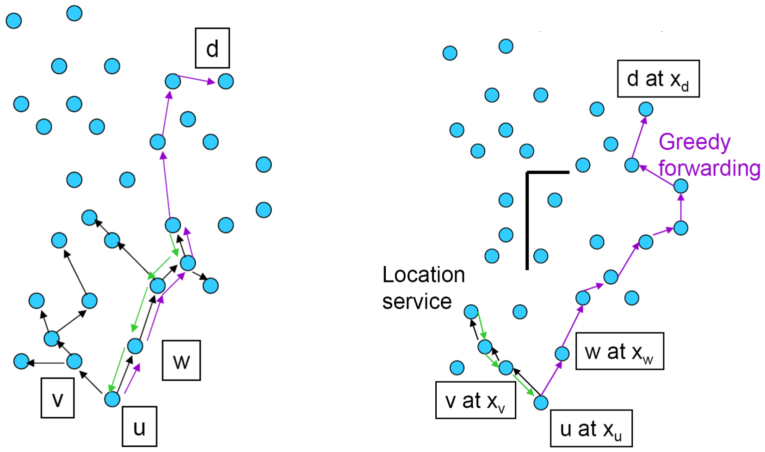
In this chapter, we investigate routing for mobile wireless networks based on virtual coordinates. More precisely, we develop a distributed algorithm to embed the network into a low dimensional space, and then perform greedy forwarding based on those virtual coordinates. The algorithms developed in this chapter are evaluated through simulations and compared to existing approaches. In Chapter 3, we will then formalize the intuition developed in this chapter by first introducing models to capture the low intrinsic dimensionality and the restricted mobility, and second by proving theoretical bounds on the stretch and the signaling traffic for routing in wireless networks. Chapter 2 is structured as follows. First, we introduce the different paradigms for routing in wireless networks. Related work is discussed in Section 2.1, and some background on graph embeddings is given in Section 2.2. In Section 2.3, we develop a random beacon-based embedding with the following features: (i) approximates well the single snapshot graph distances, (ii) gives a stable coordinate system when embedding dynamic graph topologies, and (iii) has low overhead in terms of network-wide control traffic. This embedding algorithm is then combined with a novel randomized greedy routing algorithm in Section 2.4. The idea behind the randomized routing strategy is also independently applicable to other scenarios with uncertainties in the topology or location. We give extensive simulation results to validate the properties of our algorithms in Section 2.5.

As mentioned in Chapter 1, mobile wireless networks comprise wireless devices with a limited transmission range, such as laptop computers, personal digital assistants (PDAs), cell phones, or embedded sensing and actuation devices. Such networks often rely on multi-hop communication *i.e.,* the forwarding of messages from a sender to a receiver outside the sender's radio range through intermediate nodes. One of the fundamental problems in ad hoc networking is thus the routing problem.

There are two approaches to the routing problem. The first approach relies uniquely on the topology of the network. Topology-based routing establishes an end-to-end path from a source to a destination through flooding or partial flooding of the network. By flooding, we mean that a node sends a message to all its neighbors in the network, which in turn resend the message to all their neighbors and so on until either all nodes have seen the message, or the hop count limit has been reached (partial flooding). Nodes either maintain a routing table that contains next hop and distance information for each destination [PR99], or the complete route can be stored in the header of all data packets [JMB01b]. The concept of topology based routing is illustrated in Figure 2.1(a).

The second approach exploits the geometry of the network. Nodes forward packets to neighbors geographically closer to the destination, assuming that the geographic proximity is representative of the network distance [BMSU99, KK00, KWZZ03]. Geographic routing protocols require a location service [DPH05, GV06], which can be queried to obtain the location of other nodes. For instance, a location service could work as follows. If the network area is known, every node can use a universally known hash function to map its own identifier to a position in this area. In turn, every node can send its current location to the node closest to the position obtained with the hash function. Later, when a source node intends to send a packet to a node of which it only knows the identifier, it hashes the identifier of the destination to a position in the area using the same hash function. In turn, it queries the node closest to that position to obtain the location of the destination. In Figure 2.1, we illustrate those two paradigms. Both approaches have advantages and disadvantages. The protocol overhead of topology-based routing stems from the flooding necessary for path establishment and maintenance. It is therefore very sensitive to mobility and uncertain channel environments, since even small changes in a node's neighborhood may lead to the failure of routes, which need to be reestablished. In contrast, geographic routing does not maintain state in the nodes, and forwarding decisions require only local knowledge *i.e.,* the geographic position of a node's neighbors. Topology changes that do not affect this local knowledge do not affect routing decisions, and therefore do not have to be advertised network-wide. However, geographic routing is inefficient when the network topology is not well captured by the geographic coordinates of nodes (e.g., due to a fading channel, obstacles, etc.). In such inhomogeneous networks, greedy routing towards the destination often reaches a local minimum, where no nodes with forward progress are known. Here, a recovery strategy is necessary, which requires either flooding or establishing state in nodes around the local minimum. This, together with the overhead introduced by the location service, may be more costly than the use of a topology-based routing protocol.

In this chapter, we investigate how to bridge the two paradigms. More specifically, we are interested in building a virtual coordinate system that embeds the connectivity graph in a way that is coherent with the network topology. Nodes which are close in the topology should also be close in the embedding.



(a) Topology based Routing (b) Geographic Routing

Figure 2.1: Figure 2.1(a) is an illustration of topology based routing. Typically, all-to-all shortest path are maintained either reactively, or proactively. When the source node u wants to send a message to the destination node d, it starts a scoped flood, that is a flood with a limited hop count, and if a node is reached that has a valid entry for the destination, that nodes replies to the source. The source can then forward the packet on the reverse path and routing tables can be updated. If no such node exists, the flooding radius is increased. In the worst case, the entire network is flooded and the destination itself answers to the source to setup the route. Figure 2.1(b) is an illustration of greedy geographic routing. Here, the source node u, at position $x(u)$, first greedily routes a packet to the position where the location $x(d)$ of the destination d is stored. There are different ways to obtain such a location service, one way being to use a universally known hash function to map the identifier of the destination to a position in the network area. Obviously, this implies that d must regularly send updates to the node closest to this location in order to for the location service to be up-to-date. The node that stores d's location replies to the source by forwarding the packet back greedily. Knowing $x(d)$, the source can now start forwarding packets greedily to the destination. Note that the presence of inhomogeneities such as walls can lead to dead-ends, which in turn lead to additional signaling traffic. Further, if the nodes are mobile, the location service must be updated at a sufficiently high frequency.

Desirable properties of such a coordinate system are that it allows efficient greedy routing, is robust to mobility and channel uncertainty, and is cheap to maintain.

Embeddings of general graphs (roughly speaking a distance preserving mapping of a graph metric to another metric) is a well studied topic, but few algorithms specifically for wireless graphs exist. Furthermore, ad hoc networks require an efficient *distributed* implementation of the embedding algorithm. Indeed, computing such an embedding in a centralized way would be highly inefficient in mobile networks and would lead to a high amount of signaling traffic and bottlenecks. Our main objective is to create embeddings which provide efficient routes when combined with greedy routing. This also reflects on the choice of the metric with which the wireless graph is embedded. The intuition which drives the techniques used for our embedding algorithm is the following. We believe that geometry plays an important role for long paths, whereas short paths are more subject to local perturbations. In other words, we expect that on the large scale, the hop distance in the graph must be roughly proportional to the Euclidean distance, while locally the behavior of the channel can be highly unpredictable.

2.1 Relationship to Published Works

Graph embeddings are an active area of research with many different applications [IM04]. Some of the concepts of this chapter, for example, are based on embedding algorithms used for the analysis of molecular similarities [AX03].

For networking, embeddings have been studied mainly in the context of mapping network distances in the Internet to Euclidian space [TC03,DCKM04], and relatively few embeddings exist that are specifically designed for routing in wireless ad hoc networks. The two schemes most closely related to the algorithms proposed in this chapter are the pioneering work presented in [RRP$^+$03] and the beacon vector routing (BVR) introduced in [FRZ$^+$05].

In [RRP$^+$03], a routing scheme (NoGeo) with a distributed relaxation algorithm is presented. It iteratively builds a virtual coordinate system. The algorithm assumes that a number of so-called perimeter nodes (located on the border of the network) know their real coordinates in advance. Starting from random virtual coordinates, at every iteration, non-perimeter nodes update their coordinates by averaging the coordinates of their neighbors, while the coordinates of perimeter nodes remain fixed. The scheme also includes a flooding-based mechanism to determine identity and/or coordinates of the perimeter nodes if they are not known in advance. The protocol performs well in terms of success rate of greedy routing, but the overhead to maintain perimeter nodes is very high when the topology is dynamic. The overhead also grows superlinearly with the size of the network due to the increased length of the perimeter. The algorithm critically depends on correct perimeter node information and at lower overhead may lead to a completely distorted coordinate system.

BVR [FRZ+05] is intended for routing in sensor networks. Here, the hop distances to beacon nodes directly form the virtual coordinates of a node, and the dimensionality of the coordinate space corresponds to the number of beacons. The set of beacons is randomly chosen and does not change unless a beacon fails. As in [RRP+03], greedy forwarding over the virtual coordinates is used as routing scheme. A distance metric is defined to take into account the observation that greedy routing in the direction of a beacon is more likely to lead to the destination than routing away from a beacon (explained in more detail in Section 2.5). In case of a dead-end, a small scope flooding is initiated to find a node that again provides greedy progress. The scheme is very sensitive to a bad initial choice of beacons. It also does not cope well with mobility of beacons or an uncertain channel environment, both of which lead to large shifts in the virtual coordinates. This not only results in unstable routes, but also incurs a high cost for updating a location service with the changing positions of the nodes.

There are a number of further embedding algorithms for routing in sensor networks. In [NS03], a ringed tree graph is built for data-centric information processing with coordinates that are similar to polar coordinates. Both MAP [BGJ05] and Glider [FGG+05] build a virtual coordinate system based on a tiling of the network area. The former builds a backbone structure adapted to the shape of the network topology and connects sensor nodes to the nearest backbone node through shortest paths. The latter uses Delaunay triangulation to form Voronoi cells and routes toward a so-called landmark node of the adjacent Voronoi cell that lies in the right direction. Finally, some schemes use multidimensional scaling techniques to build a coordinate system from connectivity information (e.g., [SRZF04]). All of these systems have in common that they are not designed to cope well with mobility or varying channel conditions. While the former may be less important for sensor systems (but is very important for example for vehicular ad hoc communication), the latter is an inherent property of all wireless networks.

2.2 Low-Dimensional Embeddings

In this section, we review recent results in embedding theory that motivate the beacon-based embedding algorithm presented in the next section, and provide some intuition for its design. Over the past decade, significant progress has been made in both algorithms and bounds for embeddings of (finite) metric spaces (see for example [IM04] and references therein). Given the distances $D(\cdot,\cdot)$ between n points in a metric space (X, D), the goal of the embedding is to find a mapping $x^{'} = f(x), x \in X, x^{'} \in X^{'}$ from X to another space $X^{'}$ such that for the metric $D^{'}(\cdot,\cdot)$ in $X^{'}$, for all points $x, y \in X$, the distances $D^{'}(f(x), f(y))$ do not distort $D(x, y)$ very much. More precisely, the embedding $f(\cdot)$ is said to have distortion at most c if there is a $r \in (0,\infty)$ such that for all $x, y \in X$, $rD(x, y) \leq D^{'}(f(x), f(y)) \leq crD(x, y)$. For $c = 1$, the embedding is called non-contracting. Also, typically the target space $X^{'}$ is a Euclidean space with $D^{'}$ being the l_2 norm. In such cases we can talk about the dimension of the embedding to be the dimension of $X^{'}$.

In the aforementioned framework, many graphs[1] do not admit low-distortion and low-dimensional embeddings simultaneously. There has been a significant amount of effort in classifying the distortion and dimension of specific classes of graphs (see Table 8.5.1 in [IM04]). This leads to the notion that in general graphs are not "embeddable" *i.e.,* do not admit low-distortion embeddings in low-dimensional spaces. More recently, an alternate question was posed in [KSW04], which introduced the notion of *slack* in embeddings. Slack allows a small fraction of the distances to be arbitrarily distorted, while the others are guaranteed to have a much smaller distortion. In particular, [ABC$^+$05] showed that every finite metric space can be embedded into a l_p space with constant dimension $O(\log^2(\frac{1}{\epsilon}))$ with constant distortion $O(\log(\frac{1}{\epsilon}))$, if a fraction ϵ of the distances in the original space can be arbitrarily distorted. However, the distortion may not be uniform across nodes. Therefore we need a stronger notion of slack called *uniform slack* which means that for every point $u \in X$, at most fraction ϵ of its pairwise distances can be arbitrarily distorted. Embeddings with uniform slack is also explored in [ABC$^+$05].

One of the main techniques in embeddings with slack is the use of a constant number of beacons for the embedding. Such an embedding is based on triangulation *i.e.,* reconstructing the distance between two non-beacon points from their known distances to a set of beacons. Clearly, for points that are close to each other, this can cause arbitrarily large distortion. Hence, these pairs of points are counted towards the uniform ϵ slack.

The notion of slack is inherently useful for embeddings of wireless network graphs. This is because a small number of edges suffice to transform a graph admitting a low-dimensional embedding into a graph that does not. The randomness in node locations as well as mobility and channel uncertainty (fading) can easily perturb the original geometry enough to make the graph difficult to embed completely. However, our intuition is that for wireless graphs, even though local geometry is easily perturbed through these sources of randomness, at large scales this would matter much less. Therefore, slack eliminates some of the local behavior, and allows to embed the larger-scale distances into a low-dimensional space with low-distortion. We also use this principle of slack in Section 2.4, where we develop routing algorithms.

We also note that graphs that do not possess low-dimensional embeddings can arise even without channel uncertainty. Specifically, it is possible to construct unit-disk graphs (UDG), where an edge between two nodes u and v exists if and only if $||x(u) - x(v)|| < 1$, for which no low-distortion, low-dimensional embeddings exist [MOWW04]. However, these require specific node constellations that occur only with very low probability in a random realization of node locations. Hence, these constructions are mostly of theoretical interest.

---

[1] We interchangeably use the term graph for a finite metric space.

2.2.1 Stable Dynamic Embedding of Connectivity Graph

We have discussed above the classical embedding problem of a metric space (X, D) into another (usually normed) space (X', D') through an embedding function $f : X \to X'$, and the associated metrics for the quality of the embedding (stretch, slack).[2] We now introduce a novel aspect of the embedding problem discussed in this chapter: maintaining a stable embeddings of a dynamic graph.

Indeed, the connectivity graph of a mobile wireless network changes over time because of node mobility and channel uncertainty. We can view this as a dynamic metric space $(X, D^{(t)})$, for which we would like to maintain an embedding $f^{(t)}(.)$. We define the embedded distance between x_1 and x_2 at time t as $D'^{(t)} = D'(f^{(t)}(x_1), f^{(t)}(x_2))$.

What is a good dynamic embedding? For obvious reasons, we would like the dynamic embedding $f^{(t)}(.)$ to be a faithful representation of $(X, D^{(t)})$ for every time t. However, this is not sufficient for our purposes, because it does not say anything about the evolution of the embedding over time. For example, even if the metric space $(X, D^{(t)})$ were fixed over time, the embedded coordinates might fluctuate, provided the distances $D'^{(t)}(.,.)$ remain stable. This is undesirable, for the following reason.

In our setting, although the graph changes over time, it tends to change slowly. For example, two nodes that are far apart at time t are unlikely to be very close a short time after t, and vice versa. This is a result of physical constraints on node mobility processes (nodes cannot jump from one place to another), and the fact that channels between nodes strongly depend on geography, as explained above.

Therefore, the distances $D^{(t)}$ change slowly over time, with the largest relative changes concentrated on short distances. The stable embedding problem amounts to maintaining a dynamic embedding such that (a) the instantaneous distances $D'^{(t)}(.,.)$ are close to the real distances $D^{(t)}(.,.)$ for every t, and (b) the coordinates $f^{(t)}(.)$ of the embedded space changes as slowly as possible.

A stable embedding in our context is important for the following reasons. A geo-routing algorithm has to be paired with a location service in order to be able to deliver messages to particular nodes (or information items), rather than to particular locations. A location service is essentially a distributed database that maintains the location of every node in the network. The database has to be updated when the location of a node changes. Suppose we have a node x that does not move, but whose coordinates in the embedding $f(x)$ change over time; then updates would have to be generated continually for this node, resulting in overhead. A stable embedding minimizes this overhead.

A slightly different approach eliminates the need for location updates by merging the location service into the routing protocol. In this approach, a message starts out with an imprecise estimate of the destination's location. It then refines this estimate as it travels through the network. It has been shown that it is sufficient (depending on the mobility process) that each node

---

[2] We discuss other metrics from the field of *multidimensional scaling* in [TDGW07b].

remembers when and at what location it was last a neighbor of every other node. This approach, called Last Encounter Routing (LER) [GV06], amounts to the message traveling towards past locations of the destination. If LER operates on embedded coordinates, then a stable embedding ensures that these past locations are close to the current location of the destination. Even though there is no need for location updates in LER, an unstable embedding would manifest itself through increased route cost, as the message would frequently move "in the wrong direction".

This illustrates that stable embeddings are important to minimize overhead in the context of geo-routing. Therefore, in this chapter, our goal is to develop a distributed algorithm that computes faithful and stable embeddings for slowly changing graphs.

In summary, the results on graph embeddings provide important insights for the design of the algorithms in 2.3 and 2.4. In the presence of mobility, designing a beacon management method that produces stable embeddings is challenging. Slack is an important concept for embeddings of wireless graphs since there are low probability events that can significantly deteriorate embedding performance. In Section 2.3 we explore methods that do local operations to account for such inaccuracies. There is always some degree of distortion in embeddings and routing protocols need to take this into account. To this end, we propose a randomized algorithm scheme in Section 2.4.

2.2.2 Observations on Wireless Connectivity Graph

We believe that connectivity graphs that arise from wireless networks are special, because there is some underlying geometry associated with such graphs. On the one hand, channel fading might destroy some local geometry associated with connectivity. On the other hand, nodes originate from a 2-dimensional world, and connectivity is mostly local. Hence, we suspect that geometry will play a large role in connectivity over larger distances. That is, nodes positioned far apart in the real world are bound to communicate over multiple hops. Therefore, even though one could still construct wireless connectivity graphs which do not have such properties, our experiments and intuition suggest that such configurations occur relatively rarely.

In order to understand some of these issues, we examine the problem using *multi-dimensional scaling* (MDS) which is another technique for extracting coordinates from pairwise distances widely used in statistics literature (see for instance [BG05]). Classical scaling, [BG05], is a technique used in statistics to obtain (embedded) coordinates $\mathbf{X}$ in Euclidean space of n points given only a matrix of pairwise distances[3]. Coordinates obtained with this approach minimize the sum of squared errors between the original distances and the Euclidean

---

[3] Note that the algorithm is centralized and requires knowledge of all pairwise distances. Hence, it is not suited for a distributed implementation in wireless networks. Nevertheless, it can help us get additonal insight

distances between points (which is also called the *stress function* [BG05]) *i.e.,*

$$\min_f \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left[D(v_i, v_j) - D^{'}(f(v_i), f(v_j)) \right]^2 .$$

Therefore, the criterion used to evaluate the embedding in MDS is different from that of the relative distortion criterion introduced earlier. The stretch criterion used in computer science [IM04] is a worst case distortion, in contrast to the average distortion used in MDS.

A nice property of $\mathbf{X}$ is that it is a principal axes solution *i.e.,* the variance along axes is maximized [BG05]. We are interested in this variance as we want to show that the error incurred by representing wireless connectivity graphs in low-dimensional Euclidean space is small. In other words, in an optimal least square embedding in Euclidean space, only a small number of dimensions are necessary to capture most of the variance.

Given a $n \times n$ matrix $\mathbf{D}^{(2)}$ where the (i, j)-th entry in $\mathbf{D}^{(2)}$ is given by $D^2(v_i, v_j)$, for $v_i, v_j \in \mathcal{V}$, we define a matrix $\mathbf{B}$ as,

$$\mathbf{B} = -\frac{1}{2} \mathbf{J} \mathbf{D}^{(2)} \mathbf{J}, \tag{2.1}$$

where $\mathbf{J} = \mathbf{I} - n^{-1} \mathbf{1} \mathbf{1}^{\mathbf{T}}$ and $\mathbf{1} = [1, \ldots, 1]^T$. The m-dimensional coordinate matrix of classical scaling is given by $\mathbf{X} = \mathbf{Q}_+ \Lambda_+^{\frac{1}{2}}$, where Λ_+ is the matrix containing the m largest eigenvalues and $\mathbf{Q}_+$ the corresponding eigenvectors. Note that dimensions are nested so that the $m-1$ first dimensions of a m-dimensional embedding are the same as the $m-1$ dimensions of an $m-1$-dimensional embedding

In Figure 2.2 we show how geometry plays a role in the dimensionality of connectivity graphs. We generate several $\mathcal{G}(2000, r, p)$[4] with varying average node degrees and for $p = 0.6$ and $p = 1$ (see Fig. 2.2(a) and 2.2(b) respectively). Note that since the node density is $\frac{N}{1} = N$, the average node degree is $p(N-1)\pi r^2$. We then use (2.1) to find the m-dimensional coordinate matrix associated with this topology. Assume that the eigenvalues $[\theta_1, \theta_2, ...]$ in Λ are in decreasing order. In particular, we are interested in the spectrum of the re-centered squared distance matrix $\mathbf{B}$ and how variance is distributed in the different dimensions. In Fig. 2.2, we plot $v(d) = \frac{\sum_{i=1}^{d} \theta_i}{\sum_{i=1}^{D} \theta_i}$, where D is such that θ_D is the smallest positive eigenvalue. This random experiment is repeated, and the cumulative results are presented in Figure 2.2. It can be observed that there is a considerable gap between the first and second dimensions, while

---

[4] we define a connectivity graph $\mathcal{G}(N, r, p)$, where N is the number of nodes, r is the communication radius, and p is a connection probability. First, every node i is placed uniformly at random at a position $x(i)$ on the unit square. Then, we add an edge between nodes i and j with probability p if $||x(i) - x(j)|| \leq r$ for all i, j. In this model the underlying assumption is that nodes can only communicate directly if they are located physically close to each other, and that even in this case, some losses can occur because of the random nature of wireless communications.

there are only small increments for subsequent dimensions. This indicates that most of the variance is captured by the two first dimensions. Further, this gap diminishes as we increase the communication radius. These experiments seem to suggest that wireless connectivity graphs are well representable using a small number of dimensions. Not surprisingly, when we increase the average degree and consequently the communication radius, geometry progressively plays a smaller role and the fraction of the total variance in the two first dimensions is reduced. For a very large average degree, the variance appears to be equally distributed in all dimensions. Interestingly, when we make connectivity random by setting $p = 0.6$, we reduce the variance in the two first dimensions. Intuitively, adding complexity to the channel also decreases the embeddability of the connectivity graph in a low dimensional space.

In Fig. 2.3, we show that remarkably this low dimensionality is always present, independently of the size of the network. This suggests that a smart design for a distributed embedding algorithm should exploit this property and be highly scalable. In other words, it seems that the communication overhead necessary to capture this low dimensional structure should be of $O(1)$ per node, as adding new nodes does not change the properties of the connectivity graph.

2.3 Embedding Algorithm

In this section, we describe our distributed probabilistic beaconing (PB) algorithm. We first provide some intuition on the design of the algorithm, motivated by the discussion in the previous section. We then formally define the algorithm and explain its operation through an example.

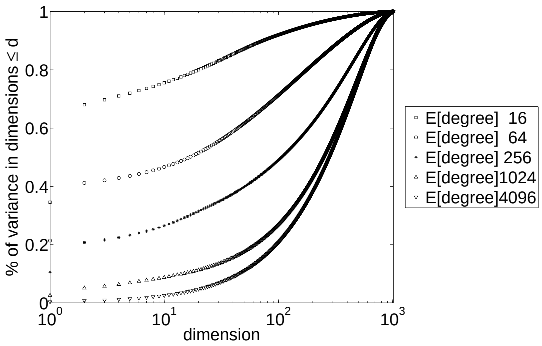
2.3.1 Embedding Heuristic

The basic idea of our algorithm is to use random beacons as anchors of an embedding. The algorithm is specifically designed to maintain a stable embedding when the graph changes over time, and to combine global beaconing with local correction operations to restore local geometry as far as possible.

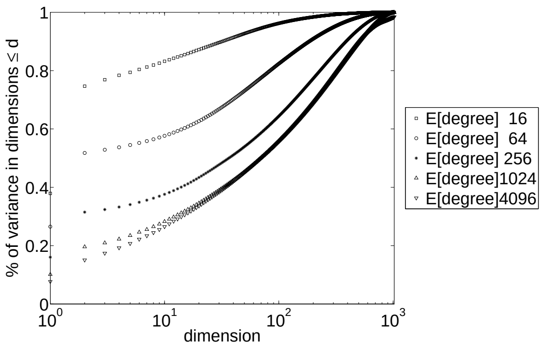
We now describe the heuristic for a node i to compute its current embedded position $x(i)$ for an embedding in M dimensions. We assume that a node i has information from a set $\mathcal{B}$ of beacons for which it knows both its graph distance h_{B_l}, $B_l \in \mathcal{B}$ and $x(B_l)$, the embedded coordinates of the beacon $B_l \in \mathcal{B}$. From this, the node attempts to find its embedded coordinates $x(i)$ using the following criterion.

$$\min_{x(i)} \sum_{B_l \in \mathcal{B}} \left[h_{B_l} - ||x(i) - x(B_l)||\right]^2 \overset{def}{=} \min_{x(i)} h(x(i)). \tag{2.2}$$

We use a heuristic to solve this optimization problem because $h(\cdot)$ is a non-convex function. Suppose node i knows its graph distance to a set of beacon nodes $\mathcal{B} = \{B_1, \ldots, B_b\}$. An iterative heuristic to solve (2.2) will update the



(a) Spectrum with $p = 0.6$



(b) Spectrum with $p = 1$

Figure 2.2: Cumulative distribution of variance in dimensions for $G(2000, r, p)$ for various average degree and probability $p = 0.6$ and $p = 1$ of link connectivity.

position $x(i)$ of node i by moving it towards or away from the beacons. This is reminiscent of stochastic proximity embedding (SPE) [Dim03].

If the wireless connectivity graph G were perfectly embeddable in two dimensions *i.e.,* if there existed a placement of the node in $\Re^2$ such that all Euclidean distances between the nodes were exactly equal to the hop distance in the graph, then, given the positions and the shortest path distances to at least three beacons, the positions of all other nodes would be uniquely determined. In reality, we expect G to be low dimensional only within some slack ϵ.

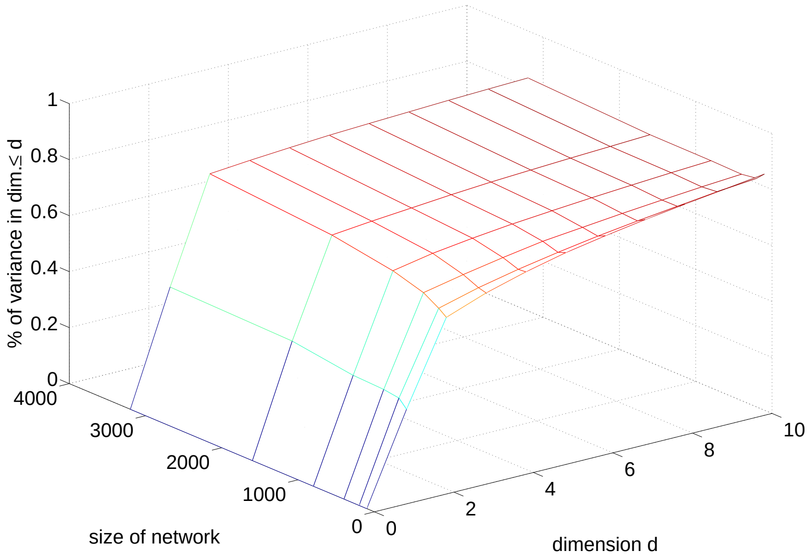


Figure 2.3: Cumulative distribution of variance in dimensions for $G(N, r, 1)$. The average degree (deg) is 16 ($r = \sqrt{\frac{deg}{p(N-1)\pi}}$) and probability $p = 1$ of link connectivity. We vary the networks size N. One can observe that the low dimensional structure of wireless connectivity graphs is independent of N

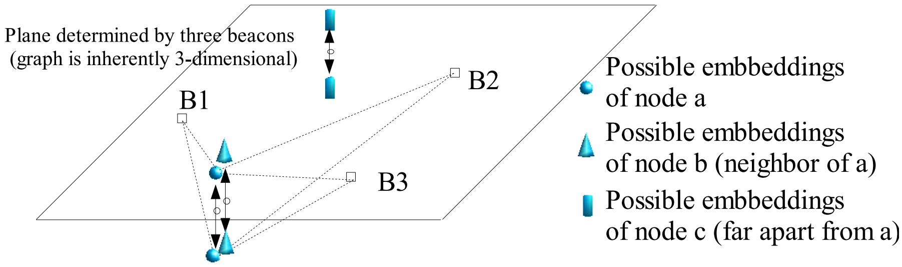

Plane determined by three beacons
(graph is inherently 3-dimensional)
B1
B2
B3
Possible embbeddings
of node a
Possible embbeddings
of node b (neighbor of a)
Possible embbeddings
of node c (far apart from a)

Figure 2.4: Nodes a and b are neighbors, while node c is far away from both. The two optimal positions of node a and b are close. In the worst case, a and b are placed on opposite sides of the plane and the relative error is approximately 1 over the distance between these optimal positions. c being placed further apart, the relative error in the distance is small. B1,B2 and B3 are beacons.

Therefore, G requires an embedding dimension of at least $M = 2$, with a small incremental benefit for higher-dimensional embeddings ($M > 2$). Conceptually, in the PB algorithm, randomly selected beacons flood the network one by one. All nodes, as soon as they get the message from a new beacon, recompute their virtual position taking into account the virtual position of the beacon, and the hop distance to the beacon. Hence, after the flooding of the third beacon, this manifests itself by the nodes clustering close to a two-dimensional hyperplane in the M-dimensional space. The relative error in the distances between non-beacon nodes is bounded by the variance in higher dimensions for nearby nodes, and becomes negligible for nodes far apart. An illustrative example where G is inherently 2-dimensional with low variance in the third dimension is shown in Figure 2.4.

2.3.2 Dealing with Dynamic Graphs

Under mobility and channel uncertainty, large scale distances remain relatively unaffected over short time scales. We exploit this slow evolution by updating the embedding in a lazy manner, but giving up on short distances. Specifically, we propose to use a sliding window mechanism to update the embedding in the face of graph dynamics. Every time the distance to a new beacon is learned, the oldest distance is thrown away. By only changing one of the beacons at a time, the coordinate system cannot change drastically as the other beacons remain fixed. Always choosing new random beacons ensures that we are not dependent on the initial choice of beacons. A new random choice at every iteration guarantees a good performance on average.

We include an additional mechanism to stabilize the embedding. To make the embedding locally consistent, nodes estimate their distances to the beacons as the average of their observed distance and the observed distances of their one-hop neighbors. The underlying idea is that when a node moves into a new neighborhood, its distance estimate should be close to the distance estimates of its new neighbors. Local coherence is especially important for geographic routing, where all forwarding decisions are local.

2.3.3 Formal Description of PB Algorithm

At every time step k, a new beacon node is randomly selected. This beacon floods the network with a control message, through which each node learns its shortest-path distance in G to the new beacon.

Our heuristic updates node i's embedded position $x(i)$, by iteratively minimizing the criterion given in (2.2) using a gradient-descent technique. Nodes know their (virtual) Euclidean distance and graph distance to beacons and consequently new beacons can be chosen with a higher probability if they are well embedded with respect to already existing beacons. Adding new beacons will push nodes out of local minima. To increase the probability that neighbors have similar coordinates (*e.g.,* to move to the same side of the plane in Figure 2.4), the input hop-distance to the algorithm is the average of a node's and its neighbors' hop-distances. More precisely, at every time k, a randomly selected beacon B_k floods the network with its virtual position $x^{(k)}(B_k)$, where the superscript indicates the time-index of the coordinate. All other nodes i in the network obtain their hop distance, or *proximity*, $P_i(k)$ to B_k in this way. We initialize $x^{(0)}(B_0)$ with a random M-dimensional vector.

Nodes have a buffer in which they store their proximities (hop-distances) to the b last beacons as well as the virtual positions of these beacons at the time they flooded. Let us call $\mathcal{B}(k)$ the set of the b last beacons at time k, and $P_i(k-l)$ the proximity of node i and beacon B_{k-l} at time $k-l$. Let $E_{ij} = ||x(i) - x(j)||$ denote the Euclidean distance between nodes i and j with positions $x(i)$ and $x(j)$ (in the virtual/embedded space) and $\mathcal{N}_i$ the one-hop neighbors of i. The probabilistic beaconing algorithm is shown in Algorithm 1. This algorithm is executed by all nodes at every time step k *i.e.,* every time a new beacon is elected and floods the network. Nodes temporarily store a vector containing the average of their distances to the beacons and the distance of their one-hop neighbors to the beacons. Then, nodes iteratively project their position on a hypersphere around every beacon of radius equivalent to the previously estimated distance $\hat{h}$ to that beacon. The input to the next iteration is the average of the projections. Figure 2.5 illustrates two iterations of the algorithm in a two dimensional space with three beacons. In practice, given the average hop-count $\hat{h}_u(k)$ of its neighborhood, node v_i attempts to minimize the criterion $h(\vec{x})$ given in Equation 2.2. The gradient of $h(\vec{x})$ with respect to $\vec{x}$ is $2\sum_{u=1}^{B} \frac{\vec{x}-b_u}{||\vec{x}-b_u||}[||\vec{x}-b_u||-\hat{h}_u]$. The iterative method proposed in Algorithm 1 is a gradient-descent method, which can be shown to go to a local minimum and therefore converge. Instead of stopping after a fixed number of steps W, we can also choose a stopping criterion such that the gradient is small enough.

2.4 Routing Algorithm

As we have explained, geometry plays a role for large distances, such that a small number of beacons are sufficient to embed at a low distortion and roughly place nodes at the correct location. To accurately embed short distances, practically every node would have to be a beacon and the dimension of the embedding would be much higher, which is not a desirable property for an embedding algorithm. We propose two local mechanisms to cope with these inconsistencies in the coordinate system.

The first mechanism is based on the principle of slack introduced in Section 2.2, which showed that the largest distortion of the distances occured in the local neighborhood. To overcome these errors, the idea is that every node can build a local routing table that is sufficiently large to overcome this local inaccuracy of the coordinate system. Therefore, the size of the slack is related to the size of such a local routing table.

Alternatively one can use Biased Random Walk (BRWalk) explained below. BRWalk is designed for coordinate system which are slightly inaccurate, such

```
input  : At time $k$, a vertex $v_i$ obtains distance $P_i(k)$ to beacon $B_k$
output: Adjusted virtual position
for $u = 0$ to $b-1$ do
    $\hat{h}_u(k) \leftarrow \frac{1}{|\mathcal{N}_i|+1}\left(\sum_{j\in\mathcal{N}_i} P_j(k-u) + P_i(k-u)\right)$;
    /* Adjust distances to beacons, averaged over
       neighborhood                                                    */
end
$x := \frac{1}{|\mathcal{N}_i|+1}\left(\sum_{j\in\mathcal{N}_i} x^{(k)}(j) + x^{(k)}(i)\right)$;
/* Starting point is center of mass of node $i$ + neighbors
   */
repeat
    $x := x + \frac{1}{b}\sum_{u=0}^{b-1}\left(\hat{h}_u(k) - ||x - x^{(k-u)}(B_{k-u})||\right)\frac{x - x^{(k-u)}(B_{k-u})}{||x - x^{(k-u)}(B_{k-u})||}$
        /* Gradient descent                                            */
until the maximum number of iterations $W$ is reached ;
$x^{(k+1)}(i) := x$;
/* Update position                                                     */
```

Algorithm 1: Probabilistic Beaconing (PB) Algorithm

as the one obtained with PB or noisy samples of real coordinates in a dense network (without large voids). It trades off path length for robustness by not trusting the coordinate system completely. In BRWalk, the next hop is chosen randomly among all neighbors. By introducing randomness into the routing decisions, we tolerate some “wrong” forwarding decisions which on one hand increase the path length but on the other hand allow to transparently avoid getting stuck in a dead-end. Nodes which are geographically closer to the destination than the current node have a higher probability of being selected as next hop, but going “backward” and loops are not excluded. In order to reduce the probability of visiting the same node several times, one can store a constant size list of the last visited nodes and reduce the probability of returning to these nodes.

Assume a source s has a packet for a destination t. For every node $j \in \mathcal{N}_s$, s computes the difference between its Euclidean distance to the destination and the Euclidean distance of node j to the destination $\Delta_j = E_{st} - E_{jt}$. Then, the probability of choosing node j as a next hop is given as $p_j = \frac{f(\Delta_j)}{\sum_{k\in\mathcal{N}_s} f(\Delta_k)}$. Additionally, if the packet can hold the identifier of the last n hops in a variable *path*, one can modify the probability of visiting a node which occurs several times in the path *i.e.,*

$$p_j = \frac{f(\Delta_j, path)}{\sum_{k\in\mathcal{N}_s} f(\Delta_k, path)} \tag{2.3}$$

One possibility is to set $f(\Delta_j, path) = \frac{e^{\alpha\Delta}}{2^m}$ where α is a parameter determining

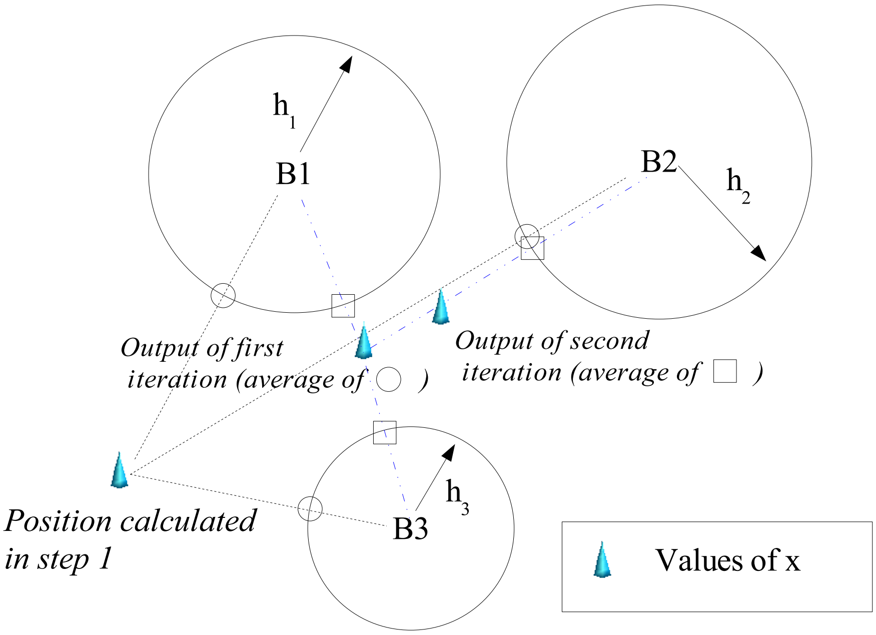

h1
B1
B2
h2
Output of first iteration (average of ○)
Output of second iteration (average of □)
h3
B3
Position calculated in step 1
Values of x

Figure 2.5: Two iterations of the PB algorithm in 2 dimensions. The node projects itself on a hypersphere of radius h_u around every beacon B_u. The input to the next iteration is the average of these projections. This implements a gradient-descent minimization of $h(\cdot)$ defined in Equation 2.2.

the "greediness" of the routing and m is the number of times node j appears in *path*. Note that when $\alpha \to 0$, the routing algorithm simply performs a random walk and that when α becomes large enough, BRWalk routing is equivalent to greedy routing (in this case the *path* is not taken into account). The next hop is a sample drawn according to the distribution given in Equation 2.3. Note that in this routing algorithm, packets need to have a time to live (TTL) field, as in the worst case, with low probability they might never reach the destination. We consider that a small percentage of lost packets is acceptable if it allows us to considerably reduce the overhead.

2.5 Simulation Results

In this section we evaluate our algorithms through a series of simulations using a custom discrete time simulator. In every round, nodes first move, then update their positions, and then communicate. A node can communicate with any other node in the network during such a round, potentially over multiple hops. We assume for simplicity that there is no packet loss at the MAC layer.

2.5.1 Experiment Design

Our embedding algorithm is designed to cope with long term fading rather than with short term fading. Consequently, we consider that nodes can move but that the channels, which are determined by the environment, do not change over time. The network model we use therefore consists of an $S \times S$ grid of locations. Every location can be occupied by none, one or several nodes. The channel existing between the locations is drawn *a priori*. Hence, if a node i occupies a location l_1 and another node j occupies a location l_2, these nodes will be directly connected through an edge (i, j) if a link exists between the two locations l_1 and l_2. In our simulation we consider that every location picks λ other locations according to an exponential distribution with mean r for the distance and at angles chosen uniformly at random around itself to connect to.[5] Note that this is not a unit disk graph model (UDG)[6], and consequently routing algorithms tailored for this particular class of graph are not applicable (*e.g.,* planarization in [BMSU99,KK00,KWZZ03]). To simulate node mobility, we use the random walk (RW) model and the random waypoint (RWP) model. When a node moves, its coordinates are rounded to the closest grid location. We compare our approach with BVR [FRZ$^+$05] and with the averaging (NoGeo) approach proposed in [RRP$^+$03]. For NoGeo, unless stated otherwise, we consider that the perimeter nodes as well as their positions are known. We made that choice as applying the "perimeter node" criteria described in [RRP$^+$03] led to a very large amount of falsely detected perimeter nodes and in turn to very poor performance with the random channel model, especially in mobile scenarios. By default, we consider 20 perimeter nodes. Unless stated otherwise, we consider a network of size $S = 30$ with $N = 1500$ nodes with $\lambda = 10$ and an expected communication range of $r = 1.5$. As a general rule, we allow an overhead of 10 messages per node to build the embeddings per round (10 averaging steps for NoGeo, 10 beacons can flood for BVR and PB). In BVR the dimension of the embedding is equivalent to the number of beacons. To make the comparison with PB meaningful, we use the same dimension of embedding for PB and BVR and 2 for NoGeo. We also use $b = 20$ for PB. The dimension of embedding M can be considered low if it is constant and $M \ll N$, where N is the number of nodes. The dimension M set to 20 by default for PB and BVR.

2.5.2 Performance Metrics

We evaluate embedding algorithms according to the following criteria:

---

[5] Channels are considered to be bidirectional, so that a location which is "chosen" by another node can have a degree higher than λ, while a node that randomly picks several times the same location might have a degree lower than λ. We bound connectivity to λ locations for simplicity, and verified that the results are equivalent to an exponential distribution over all points.

[6] In a UDG model, nodes i and j with positions $x(i)$ and $x(j)$ respectively are connected if and only if $||x(i) - x(j)|| \leq r$, for a fixed communication radius r.

Distortion

Given two nodes i and j with virtual coordinates $x(i)$ and $x(j)$, we define the multiplicative distortion as $\frac{||x(i)-x(j)||}{r_{ij}}$ where r_{ij} denotes the shortest path distance in the connectivity graph between i and j.

Greedy Routing Success Rate (GSR)

The fraction of packets that reach their destination by making only local forwarding decisions based on the coordinates of the nodes in the neighborhood and the position of the destination. For PB and NoGeo, we use classic greedy routing which forwards a packet to the neighbor closest in Euclidean distance to the destination. For BVR, we use both greedy routing and the routing algorithm proposed in [FRZ$^+$05], here called "BVR greedy" and "BVR", respectively. For the latter, the distance between two nodes p and d is given by $A\delta^+ + \delta^-$, where $\delta^+(p,d) = \sum max(p_i - d_i, 0)$ and $\delta^-(p,d) = \sum max(d_i - p_i, 0)$. Index i corresponds to the i^{st} coordinate. As in [FRZ$^+$05], we set weight $A = 10$ and take all beacons into account. The neighbor that minimizes this distance function is chosen as a next hop.

Path Stretch

The path stretch is the ratio between the actual number of hops a data packet traveled from a source to the destination and the shortest path distance in hops between these two nodes. We only consider successful communication.

Communication Overhead

We consider as overhead all packets that are not data packets. This includes the packets flooded by beacon nodes as well as the packets used to build local routing tables and the packets used in ring search.

Virtual speed

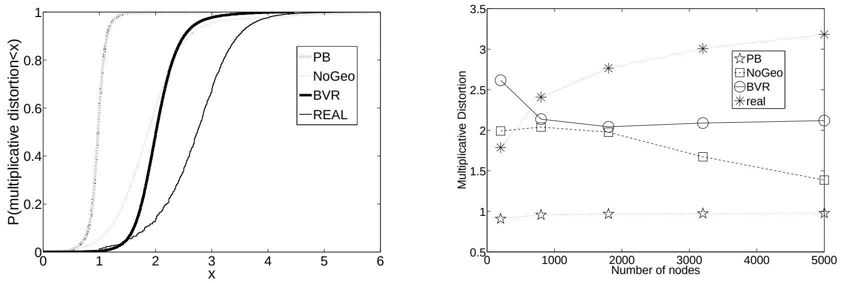
This metric captures how fast nodes move in the virtual coordinate space, *i.e.,* the average Euclidean distance between the virtual coordinates of nodes from one round to the next.

2.5.3 Static Networks

In this section, we investigate the performance of PB in static networks both in terms of embedding quality and routing efficiency.

Quality of Embedding

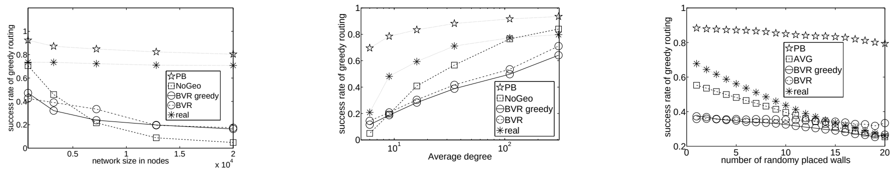
In Figure 2.6(a), we show the empirical cumulative distribution function (ecdf) of the multiplicative distortion in a fixed network size. It can be seen that it is low with PB. This indicates that PB can efficiently capture the inherently low dimensional structure of wireless connectivity graphs. Further, a small number of beacons suffice to this end. One can also point out the fact that the slope of the cumulative distortion curve with PB is almost vertical which indicates a lower variance. It is interesting to note that the distortion of BVR is high, since nodes are spread out in all dimensions. Due to the randomness of the channel, there can also be highly distorted distances with the real coordinate system (*e.g.,* if nodes located in neighboring locations are not connected directly). With NoGeo, the averaging procedure can place nodes arbitrarily close or far apart so that a fraction of distances are highly distorted. As shown in Fig. 2.6(b), the mean multiplicative distortion does not appear to grow considerably with the size of the network. In addition, it stays very close to 1 with PB, which tends to indicate that even the additive distortion is small. A direct consequence will be that geographic routing performs well on top of a virtual coordinate system built with PB, independently of the size of the network.



(a) Multiplicative distortion

(b) Mean multiplicative distortion versus network size

Figure 2.6: Cumulative distribution function of multiplicative distortion for 2000 nodes and mean distortion as a function of number of nodes. When we increase the number of nodes, we maintain a constant density of 2 nodes per grid location.



(a) Scalability

(b) Density

(c) Inhomogeneity

Figure 2.7: GSR as a function of network size, node density and number of obstacles (randomly placed straight walls of length 6). The overhead allowed to build the embedding is set to 20 messages per node.

Quality of Routing

We will first investigate the performance of greedy routing on top of virtual coordinate systems in a static setting. In particular, we will focus on the performance of the algorithms in networks of increasing size, networks of increasing node density and in inhomogeneous network topologies. We will also investigate local optimization mechanisms to improve the quality of routing.

Scalability, Density, and Inhomogeneity Increasing the size of the network while maintaining a constant density of 2 nodes per grid location has the effect of introducing larger distances in the topology. The overhead per node allowed to build the embedding is kept constant. A consequence is that in PB and BVR, the beacons are spread out. We show in Figure 2.7(a) that this affects the GSR of PB only marginally. On the other hand, a clear effect can be seen with NoGeo as the node positions depend on the position of perimeter nodes and the averaging takes more time to reach the center of the network. A direct consequence of the way the BVR embedding is built is that is is easier to route toward beacons. Indeed, one can observe that the GSR decreases remarkably with this approach when we increase the network size. A similar phenomenon can be observed when the network area is reduced and the number of nodes is kept constant. Figure 2.7(b) shows that NoGeo and also BVR are severely affected by changes in the network diameter, while PB is relatively unaffected.

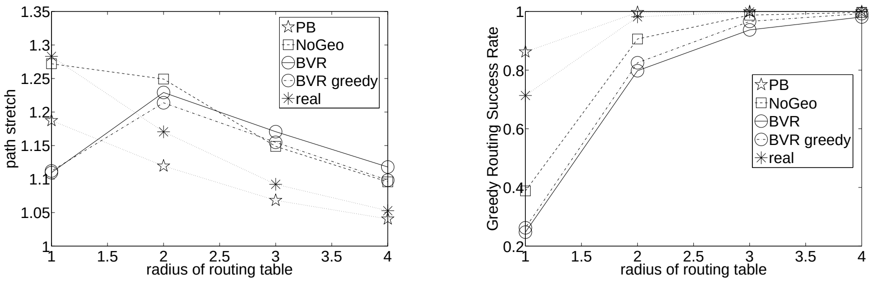
Increasing the network size also creates voids in the topology as not all grid locations are occupied anymore, which reduces the GSR of real coordinates. This effect can also be seen in topologies with obstacles (see Figure 2.7(c)).

Note that the GSRs of NoGeo and BVR are fairly low since we limit all approaches to the same overhead to build the embedding and the convergence speed of PB is higher. Our simulations have shown that in order to reach the same GSR as PB, NoGeo and BVR need an overhead of up to 100 messages per node in this setting.

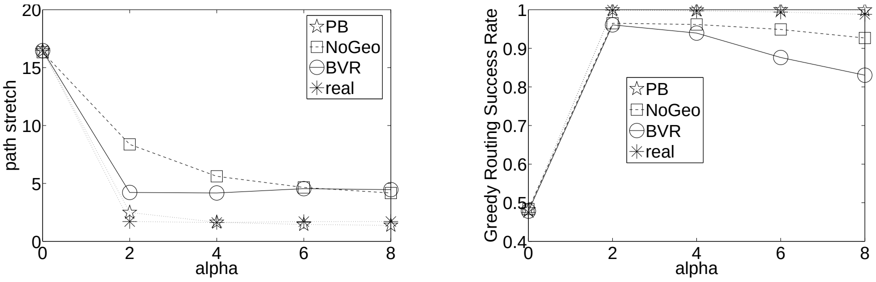
Local Optimizations As explained in 2.2, small distances are hard to embed.

As explained in Section 2.4, allowing nodes to build local routing table of growing scope can increase the GSR and reduce the path stretch remarkably as shown in Fig. 2.8(b) and 2.8(a), but the overhead to build the routing tables increases likewise.

BRWalk allows to trade path length for overhead. In Figure 2.8, we show that for a sufficiently high value of the parameter α (see Section 2.4), the stretch can be as low as 2 while the GSR reaches 100%, which is similar to the one obtained with local routing tables at no cost in terms of overhead. This shows that when a good next hop is available that is much closer to the destination than the current node, this node should be chosen. In other words, if we make a big progress in Euclidean space, we should trust the embedding. On the other hand, when no such node exists, a next hop should be chosen randomly. This result also suggests that there are small errors in the embedding. The ordering with respect to a target node of nodes embedded nearby can be perturbed. In turn, this phenomenon leads to wrong forwarding decision. Introducing randomness in the forwarding decisions appears to be an efficient way to mitigate the effects of such disorderings.



(a) Path stretch (b) Greedy success rate



(c) Path stretch (d) Greedy success rate

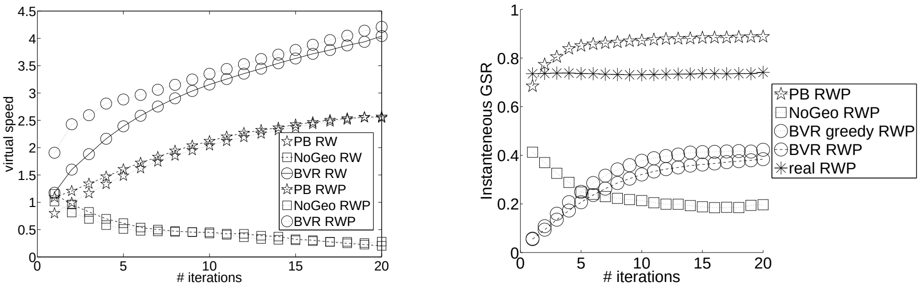
Figure 2.8: GSR and path stretch for BRWalk routing and forwarding with local routing tables. α captures the randomness of the forwarding in BRWalk. A greater value for α means less randomness. In this experiment, packets have a memory of 20 hops, as explained in Section 2.4.

2.5.4 Mobile Networks

Here, we study the behavior of embedding algorithms when the wireless connectivity graph is dynamic because of node mobility.

Quality of Embedding

In mobile networks, we measure the quality of embedding by looking at the average virtual speed of nodes in the virtual coordinate system. Only a certain amount of "fresh" distance information is injected per round. In this case, the embedding is partly built based on outdated distance information. The virtual speed with PB is of the same order as the real speed of the nodes. The sliding window mechanism mitigates the effect of injecting new distances while the averaging mechanism mitigates the local incoherence in the coordinate system. Again, we assume that for NoGeo perimeter nodes are given, since the perimeter node detection procedure in [RRP$^+$03] (a node is a perimeter node if it is further away from a beacon node than all of its two hop neighbors) leads to a large number of false positives and ultimately to the collapse of the coordinate system. In Figure 2.9(a), we show the average virtual speed of nodes when the nodes move according to the RW and RWP model with a speed of 1.



(a) Virtual Speed (b) Greedy Success Rate

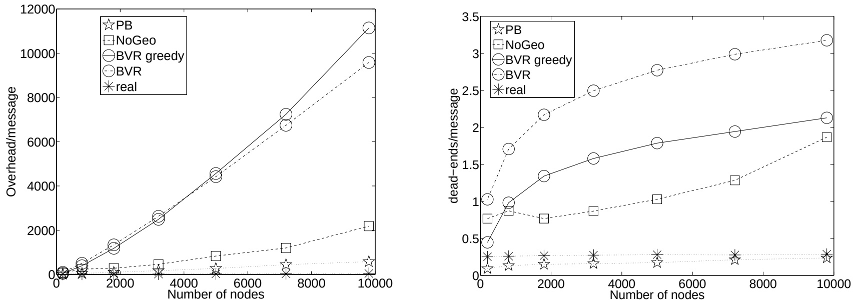
Figure 2.9: Virtual Speed and GSR under mobility as a function of the allowed number of iterations to update the coordinate system between every move

Quality of Routing

Second, we analyze the quality of routing in a dynamic setting.

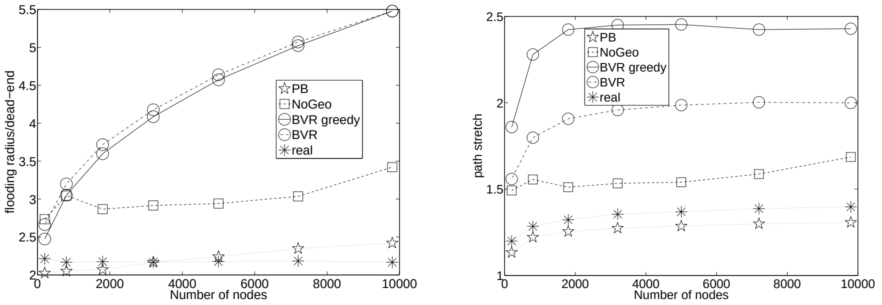
Instantaneous GSR We investigate the performance of PB when we limit the number of iterations to update the algorithm in every round. Figure 2.9(b) shows that as few as 5 iterations are sufficient to obtain a high GSR. For the sake of clarity we only show the results for the RWP as the RW results are very close. It is remarkable that even under mobility a higher GSR than with real coordinates can be obtained. There is an interesting trade-off between stability and GSR. The more fresh information is added to the embedding, the more it moves but the better the GSR. In a static environment, BVR is more stable since the beacon nodes never change, but the GSR of PB can still be up to two to three times higher than that of BVR, depending on the number of beacons. In a mobile environment, however, PB is both more stable and more efficient in terms of GSR. Note that the GSR of NoGeo decreases because new perimeter nodes are selected continuously, and the coordinate averaging does not manage to propagate through the network sufficiently fast to keep up with the new information.

Reliable routing We now investigate reliable routing under mobility. When a packet is stuck with greedy forwarding, a ring search is started until a node



(a) Overhead per packet

(b) Dead-ends per packet



(c) Flooding radius per dead-end

(d) Path stretch

Figure 2.10: Reliable routing under mobility. Nodes move according to the RWP model with speed 1 and pause time 0. The are on average 2 nodes per grid location and we maintain this density fixed while we increase the number of nodes. Nodes are allowed an overhead of 5 messages to update the embedding before they move again (5 iterations). The dimension of embedding is 20 for BVR and PB and 2 for NoGeo

closer to the destination is found for NoGeo and PB. For BVR, we use the recovery strategy proposed in [FRZ$^+$05] which routes a packet toward the beacon closest to the destination. If along the way, the packet reaches a node which is closer to the destination than the dead-end, greedy mode is resumed. Otherwise, a ring search is started from the beacon. In Fig. 2.10, we are interested in how the communication overhead to achieve reliable communication scales with the size of the network. We allow only a fixed overhead per node to update the embedding in every round, and consequently the embeddings will be based on outdated distance information. In Fig. 2.10(a), it can be seen that reliable communications are less costly in terms of overhead with PB than with the other approaches. Note that the overhead with real coordinates remains low as there is no additional cost to update the real coordinate system. The gap between the different schemes grows as the size of the network increases. Fig. 2.10(b) and Fig. 2.10(c) give us some insight. In the former, it can be seen that the number of dead-ends per packet sent remains very low with PB. In the latter, it is shown that when such a dead-end occurs recovery is cheap. Indeed, on average a node only needs to search its two hop neighborhood to find a suitable next hop. With the other approaches, the number of dead-ends per packet increases with network size as well as the search radius necessary to find a next hop. It is also worthwhile to note that the path stretch with PB remains more or less constant. These are indicators that the quality of the embedding and its stability do only scale well with PB.

Location service Overall, our simulations show that the embeddings built with PB are sufficiently stable to be used with Last Encounter Routing (LER) [GV06], resulting in a complete solution for routing in dynamic wireless ad hoc networks with low overhead. As expected, PB is particularly efficient compared to real coordinates in inhomogeneous topologies. Indeed, our experiments have shown that in settings with obstacles the overhead to route a packet with a PB embedding and LER (including the overhead to build the embedding) remains lower than the overhead necessary to route a packet with LER on top of real coordinates. This remains true when we increase the network size. We refer the reader to [TDGW07b] for a more in depth explanation of LER on top of PB. Note that when the topology is fixed, all embeddings oscillate very little and are consequently also adapted for classical location services (LS). Indeed, location services updates are in general triggered when a node has moved a certain distance.

2.6 Concluding Remarks

In this work we present a novel embedding algorithm for greedy routing on virtual coordinates that is robust to network dynamics and random channel conditions. Its key feature is the intricate combination of local information (averaging) and global information (probabilistic beaconing) in an iterative, distributed manner. We give some intuition why these connectivity graphs are inherently well embeddable even in the face of mobility. The proposed algorithm significantly outperforms existing embeddings in terms of success rate of greedy routing, convergence and distortion of the embedding, and overhead in static as well as dynamic environments. We further show that some degree of randomness in the routing decisions alleviates the effects of local inaccuracies in the positions of the nodes. In Chapter 3, we will formalize the intuition developed in this chapter.

Hierarchical Routing in Dynamic Ad hoc Networks

3

As we have seen in the previous chapter, a major challenge in the design of wireless ad hoc networks is the need for distributed routing algorithms that consume a minimal amount of network resources. Clearly, this is particularly important in dynamic networks, where the topology can change over time, and therefore routing tables must be updated frequently. Such updates incur control traffic, which consumes bandwidth and power. Hence, we concluded that it was natural to ask whether there exist low-overhead schemes for dynamic wireless networks that could produce and maintain efficient routes. In this chapter, we again consider dynamically changing connectivity graphs that arise in wireless networks. Our performance metric for the algorithms is the average signaling overhead incurred over a long time-scale when the topology changes continuously. Our goal is again to design a routing algorithm which can cope with such variations in topology. We intend to maintain efficient routes from any source to any destination node, for each instantiation[1] of the connectivity graph. By efficient, we mean that we want to guarantee that the route is within a (small) constant factor, called *stretch* of the shortest path length. However, in this chapter, we attack the problem in a different way. Recall that in Chapter 2, we embedded the network in a low-dimensional space, so that we could perform greedy routing on top of the virtual coordinate system. In this chapter, we try and get rid of the need for a separate location service. In order to route to a destination, we need only the identity of the destination and not its address *i.e.,* the control traffic to maintain the mapping between node identity and address/location is incorporated into the overhead. Therefore, in the wireless routing terminology, we have included the "location service" in the control signaling requirement, and therefore hope to characterize the complete

---

[1] We assume inherently that the round-trip time (RTT) of a packet from source to destination is much smaller than the time-scale of topology change.

overhead needed to maintain efficient routes. In contrast to the approach in Chapter 2 that mapped the network to a low dimensional space independently of its intrinsic dimension, we try here to develop an approach that adapts to this intrinsic dimension. The results presented in Chapter 2 were based on simulations of heuristic routing and embedding algorithms. While these approaches appear to be very promising and gave us an intuition about how to deal with dynamic wireless networks, the difficulty to analyze the performance formally is somewhat limiting. On the other hand, the new approach presented in this chapter, as well as the theoretical analysis of the properties of wireless networks, allows us to derive asymptotic bounds on the signaling overhead and the stretch. Further, the results presented in this chapter can also give us insights about why the embedding algorithms of Chapter 2 performed so well.

In order to develop and analyze the routing algorithms we utilize the underlying geometric properties of the connectivity graphs which arise in wireless networks. This geometric property is captured by the *doubling dimension* of the connectivity graph. A graph induces a metric space by considering the shortest path distance between nodes as the metric distance. The doubling dimension of a metric space is the number of balls of radius R needed to cover a ball of radius $2R$. For example a low-dimensional Euclidean space has a low doubling dimension as will be illustrated in Section 3.2. A metric space having a low (constant independent of the cardinality of the metric space) doubling dimension is called "doubling". We show that several wireless network graphs (under conditions given in Section 3.2) are doubling and therefore enable the design and analysis of hierarchical routing strategies. In particular, it is not necessary to have uniformly distributed nodes with geometric connectivity for the doubling property to hold, as illustrated in Figure 3.2 in Section 3.2. Therefore, the doubling property has the potential to enable us to design and analyze algorithms for a general class of wireless networks. Moreover, for a large class of mobility models, the sequence of graphs arising due to topology changes are all doubling (for specific wireless network models). Since there are only "local" connectivity changes due to mobility, there is a smooth transition between these doubling graphs. We can utilize the locality of topology changes to develop lazy updates methods to reduce signaling overhead.

We show that several important wireless network models produce connectivity graphs that are doubling. In particular, we show that the geometric random graph with connectivity radius growing as $\sqrt{\log n}$ with network size n, the fully connected regime of the dense or extended wireless network with signal-to-interference-plus-noise ratio (SINR) threshold connectivity, and some examples of networks with obstacles and non-homogeneous node distribution are doubling. We define a sequence of wireless connectivity graphs to be *smooth* if each of the graphs is doubling and the shortest path distance between two nodes in the graph changes smoothly (defined in Section 3.2). This defines mild regularity conditions on the mobility model.

Our main results in this chapter are the following. (i) For smooth geometric sequence of connectivity graphs, we develop a routing strategy based on a hierarchical set of beacons with scoped flooding. We also maintain cluster membership for these beacons in a lazy manner adapted to the mobility model and doubling dimension. (ii) We develop a worst-case analysis of the routing algorithm in terms of total routing overhead and route quality (stretch). We show that we can maintain constant stretch routes while having an average network-wide traffic overhead of $O(n \log^2 n)$ bits per mobility time step. The load-balanced algorithm would require $O(\log^3 n)$ bits per node, per mobility time. Through numerics we show that the theoretically obtained worst-case constants are conservative.

3.1 Relationship to Published Works

Routing in wireless networks has been a rich area of inquiry over the past decade or more. In the introduction of Chapter 2, we gave an overview of the two main paradigms for routing in wireless ad hoc networks: geographic routing and topology based routing. In summary, geographic routing algorithms (see for instance [KK00] and references therein) exploit the inherent geometry of wireless networks, and base routing decisions directly on the Euclidean coordinates of nodes. Their performance depends on how well the Euclidean coordinate system captures the actual connectivity graph, and these approaches can therefore fail in the presence of node or channel inhomogeneity (like in Figure 3.2 in Section 3.2). Another important, but often overlooked, issue with geo-routing is that geographical positions of the nodes need to be stored and continuously updated in a distributed database in the network, to allow sources of messages to determine the current position of the destination. This database is called a *location service* (see for instance [LJDC$^+$00]) and must be regularly updated so that source nodes can query it. Location services typically rely on some a-priori knowledge of the geographical boundaries of the network. This is necessary because these approaches typically establish a correspondence (for example, through a hash function) between a node identifier and one or several geographical locations where location information about that node is maintained. An important feature of our work is that we consider the total overhead incurred by the update and lookup operations of the location service, and the overhead of the routing algorithm itself.

Topology based routing schemes (see [PR97] and [JMB01a]) do not utilize the underlying geometry of wireless connectivity graphs, but instead compute routes based directly on that graph. As explained in Chapter 2, to reduce overhead, most of these schemes only establish routes *on demand* through a route discovery operation, rather than continuously maintaining a route between every pair of destinations; in this respect, they differ significantly from their counterparts for the wired Internet (such as OSPF, IS-IS, and RIP). Recently established routes are cached in order to allow their reuse by future messages. In distance-vector based approaches (e.g., [PR97]), this cached state resides in the intermediate nodes that are part of a route, whereas in source-routing approaches (e.g., [JMB01a]), the cached state resides in the source of a route. Despite such optimizations, topology-based approaches suffer from the large overhead of frequent route discovery operations in large and dynamic networks. This issue was, in fact, the reason why geo-routing approaches have reached prominence.

Two schemes that utilize the underlying geometry of graphs are the works presented in [RRP$^+$03] (referred to as "NoGeo" in Chapter 2), and the beacon vector routing (BVR) introduced in [FRZ$^+$05]. We used those two schemes as performance benchmarks in Chapter 2. These schemes were designed for *static* wireless networks[2]. Both these schemes are heuristics which build a virtual coordinate system. Geographic routing on top of these schemes was shown to work well through numerics, when compared to routing on top of real coordinates. However, they utilize an external addressing scheme to make a correspondence between addresses and names. In [TDGW07a] (the work presented in Chapter 2), we studied routing on dynamic networks using a virtual coordinate system. For large scale dynamic wireless networks, these heuristics pointed to significant advantages to using some geometric properties for routing and addressing. These results motivated the questions studied in this chapter.

There has been a vast amount of theoretical research on efficient routing schemes in wired (*i.e.,* static) networks (see for example [Gav01]). Most of this work has been focused on the trade-off of memory (routing table size) and routing stretch. There are two main variants of such routing schemes (i) *labeled* (or *addressed*) routing schemes, where the nodes can be assigned addresses so as to reflect topological information; (ii) *named* routing, where nodes have arbitrary names, and as part of the routing, the location (or address) of the destination needs to be obtained (similar to a location service). This examines the important question of how the node addresses need to be published in the network. Routing in graphs with finite doubling dimension has been of recent interest (see [KRX06], and references therein). In particular [Tal04] showed that one could get constant stretch routing with small routing table sizes for doubling metric spaces, when we use labeled routing. This result was improved to make routing table sizes smaller in [CGMZ05]. The problem of named routing over graphs with small doubling dimension has been studied in [KRX06] and [AGGM06], and references therein. To the best of our knowledge, there has been no prior work on *dynamic* graphs over doubling metric spaces and on control traffic overhead. It is worth pointing out that there is no direct correspondence between control traffic and memory. Bounds on memory do not take into account the amount of information which needs to be sent around in the network in order to build routing tables. A good illustration is the computation of the shortest path between two nodes u and v in a graph. While it is sufficient for every node on the path between these two nodes to have one entry for v (of roughly $\log n$ bits *i.e.,* the name of the next hop), computing that shortest path requires a breadth first search of the communication graph and leads to a control traffic overhead of $O(n \log n)$ bits.

---

[2] Our simulations (see Chapter 2, Figure 2.10) confirmed that their performance degrades substantially under mobility

3.2 Models and Definitions

A wireless network consists of a set of n nodes spread across a geographic area in the two-dimensional plane. We model the network region as the square area $[0, \sqrt{n}) \times [0, \sqrt{n})$. The n nodes move randomly in this area and we denote by $x^{(t)}(u)$ the position of node u at time t. The connectivity between two nodes is represented by an edge on the connectivity graph $\mathcal{G}_n^{(t)}$ if they can communicate *directly* over the wireless channel. The connectivity between two nodes depends on the distance between the two nodes (and could also depend on the presence of other nodes, see Section 3.2.2). We consider that when a node u transmits on the wireless channel, it broadcasts to all its neighbors in the connectivity graph $\mathcal{G}_n^{(t)}$. Consequently, one transmission of a packet is sufficient for all direct neighbors to receive that packet. To make the notation lighter, we will only add the dependence on time if it is necessary to avoid confusion. The distance $d^{(t)}(u, v)$ between nodes u and v is the shortest path distance between these nodes in $\mathcal{G}_n^{(t)}$. Note that $d(.,.)$ is a metric on $\mathcal{G}_n^{(t)}$, *i.e.,* the distance between a node and itself is zero, the distance function is symmetric and the triangle inequality applies. We will now define a *ball* of radius R around a node u. It is simply the set of nodes within distance R of u. More formally, we can define it more generally for any metric space as follows:

Definition 3.1. *A Ball $\mathcal{B}_R^{(t)}(u)$ around node u at time t in a metric space $\mathcal{X}$ is the set $\left\{v \in \mathcal{X} | d^{(t)}(u, v) \leq R\right\}$.*

In order to bound the control traffic overhead, we will recursively subdivide the connectivity graph into balls. It will be crucial for us to bound the number of balls of radius R necessary to cover a ball of radius $2R$ around some node u. In other words, we want to find the smallest number of nodes v_i such that all nodes within $2R$ of u are also within R of some node v_i. The notion of doubling dimension of a metric space captures this idea.

Definition 3.2. *The doubling dimension of a metric space $\mathcal{X}$ is the smallest α such that any ball of radius $2R$ can be covered by at most α balls of radius R, for all $R \geq \min_{(u,v)} d(u, v)$* i.e., *$\forall u \in X \ \exists \ S_u \subseteq \mathcal{X}, |S_u| \leq \alpha$ and*

$$\mathcal{B}_{2R}^{(t)}(u) \subseteq \bigcup_{j \in S_u} \mathcal{B}_R^{(t)}(j)$$

Moreover, if α is a constant, we have the following definition:

Definition 3.3 (Doubling metric space)**.** *We say a metric space $\mathcal{X}$ is* doubling *if its doubling dimension is a constant.*

A good way to illustrate and understand the concept of doubling dimension and doubling metric space is to look at the metric space defined by a set of points $\mathcal{X}$ in $\mathbb{R}^2$ with the Euclidean distance. A ball of radius $2R$ around a point x will simply be a disc of radius $2R$ around this point. To cover this disc, we will select a set of points such that all the surface is covered by the corresponding set of discs of radius R. Note that the number of discs required will not depend on R, and consequently this metric space would be *doubling* (see Figure 3.1).

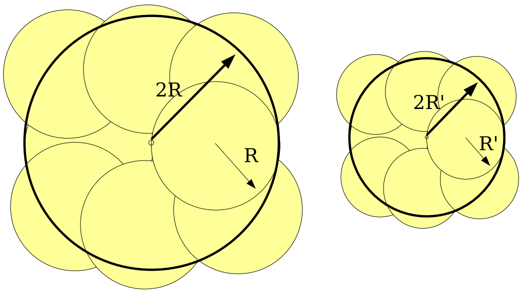


Figure 3.1: The metric space defined by a set of points in $\mathbb{R}^2$ and the Euclidean distance is doubling. Indeed, we can cover a disc of radius $2R$ by a constant (8 in this case) number of discs of radius R, whatever the value of R. The number of small balls required to cover the large ball depends on the "volume" of the balls, and not on their cardinality.

In Section 3.2.1, we describe the geometric random graph model, which will be the canonical model we will use to illustrate the ideas of the chapter. We also give an example of a non-homogeneous network to which our results can be applied. In Section 3.2.2, we will develop the model where connectivity is determined by the SINR, and we have uniform transmit power and full connectivity. We give the requirements for the mobility model to result in a *smooth* sequence of wireless network graphs in Section 3.2.3. We state the underlying assumptions and give a table of notations in Section 3.2.4.

3.2.1 Geometric random graph

We denote the geometric random graph by $\mathcal{G}(n, r_n)$ and define its connectivity as follows.

Definition 3.4. *A random geometric graph* $\mathcal{G}(n, r_n)$ *has an unweighted edge between nodes* u *and* v *if and only if* $||x(u) - x(v)|| < r_n$*, where* $\{x(u)\}$ *are chosen independently and uniformly in* $[0, \sqrt{n}) \times [0, \sqrt{n})$.

In this chapter we will be interested in fully connected geometric random graphs, and therefore focus on the case $r_n > \sqrt{\log n}$ [GK98]. As a natural extension, we can also define a sequence of random graphs $\mathcal{G}^{(t)}(n, r_n)$ with an unweighted edge between u and v at time t if $||x^{(t)}(u) - x^{(t)}(v)|| < r_n$. Whether each graph in the sequence $\mathcal{G}^{(t)}(n, r_n)$ corresponds to a random geometric graph as in Definition 3.4, depends on the mobility model for the nodes. We discuss this in more detail in Section 3.2.3.

In Figure 3.2, we illustrate a non-homogeneous random network where connectivity is not completely geometric as in Definition 3.4. An obstacle prevents communication between neighboring nodes, and therefore illustrates the complexities of wireless network connectivity. This example is revisited in Section 3.3, where we show that though this connectivity graph is more complicated than $\mathcal{G}(n, r_n)$, it is still doubling, and therefore the algorithms developed in this chapter are applicable. This illustrates the advantage of our approach to network modeling.

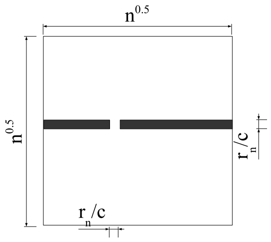


Figure 3.2: n nodes are distributed uniformly at random on a square area of side $\sqrt{n}$. A wall of width r_n/c is added, which only has a small hole in the middle. Again, we assume $r_n > \sqrt{\log n}$. Nodes cannot communicate through the wall. Finally, we remove the nodes below the wall, which leads to an inhomogeneous node distribution.

3.2.2 SINR full connectivity

Since the wireless channel is a shared medium, the transmissions between nodes interfere with each other. However, the signal strength decays as a function of the distance traveled, and therefore we can define the SINR for transmission from node u to v as,

$$\mathrm{SINR} = \frac{P_n ||x(u) - x(v)||^{-\beta}}{N_0 + \sum_{w \neq u,v} P ||x(w) - x(v)||^{-\beta}}, \tag{3.1}$$

where β is a distance loss (decay) parameter depending on the propagation environment, P_n is the common transmit power of the nodes and N_0 is the noise power. We can of course easily adapt this to have power control for the nodes. A transmission is successful if the SINR is above some constant threshold value ς. For static nodes, just as in the case of geometric random graph, we assume that the node locations $\{x(u)\}$ are chosen independently and uniformly in $[0, \sqrt{n}) \times [0, \sqrt{n})$. This model for wireless networks has been extensively studied in the literature (see [GK00, KV02]). The authors base their analysis of the capacity of wireless networks on a TDMA scheme for the SINR connectivity model of (3.1). We argue here that the structure of the resulting connectivity graph is identical to that of $\mathcal{G}(n, r_n)$, for $r_n > \sqrt{\log n}$. Therefore, the results we prove for $\mathcal{G}(n, r_n)$, would also be applicable to such graphs. In practice, it is a non-trivial task to design a distributed scheduling protocol (MAC layer protocol) that mimics the behavior of this TDMA scheduler. However, these MAC layer implementation issues are beyond the scope of this document (see for instance [MSZ06]). We only make the argument here that the connectivity graph resulting from such a TDMA scheme would yield the same behavior as a $\mathcal{G}(n, r_n)$.

We will subdivide the network into small squares of side $s_n = \frac{r_n}{c}$. We need to show that if two nodes u and v are in neighboring small squares (and so have the guarantee that they can communicate under the $\mathcal{G}(n, r_n)$ model as we will see in the sequel), then there exists a TDMA scheme that allows them to communicate under the SINR connectivity model of (3.1). If this is the case, then we can apply the same proof techniques for both models. We let the maximum transmission power grow in the same way as we did for the $\mathcal{G}(n, r_n)$ model[3] *i.e.* $P_n \leq (N_o \varsigma r_n)^\beta$. Additionally, we want to design a TDMA scheme such that the capacity of all links is at least $O(\frac{1}{\log n})$ [*bits/sec*]. It can be shown (see [RS98]) that every small square contains at most $O(\log n)$ nodes with high probability. Hence, we ask that the traffic can flow at constant rate independent of n between neighboring small squares, and that each node is treated equally. Note that this requirement is very similar to the scheme proposed in [GK00] in which one node per small square can transmit at constant rate to any neighboring square[4].

Theorem 3.1. *There exists a TDMA scheme such that all nodes can communicate with any node located in a neighboring small square at a rate of* $O(\frac{1}{\log(n)})$ [*bits/sec*]. *Hence, the aggregate traffic can flow between neighboring small squares at a constant rate independent of* n.

Proof. We take a coordinate system, and label each square with two integer coordinates. Then we take an integer k, and consider the subset of squares whose two coordinates are a multiple of k (see Figure 3.3). By translation, we can construct k^2 disjoint equivalent subsets. This allows us to build the following TDMA scheme: we define k^2 time slots, during which all nodes from a particular subset are allowed to transmit for the same duration of $O(\frac{1}{\log n})$ seconds. Each small square contains at least one and at most $O(\log n)$ nodes w.h.p. (see [RS98] and the proof of Theorem 3.3). We assume also that at most one node per square transmits at the same time, and that they all transmit with the same power P_n. Let us consider one particular square. We suppose that the transmitter in this square transmits towards a destination located in a square at distance at most 1. We compute the signal-to-interference ratio at the receiver. First, we choose the number of time slots k^2 as follows: $k = 4$. To

---

[3] Note that the $\mathcal{G}(n, r_n)$ model corresponds to the SNIR model without interferences. Indeed, if we remove interferences, two nodes can communicate whenever $\frac{P_n ||x(u)-x(v)||^{-\beta}}{N_0} > \varsigma$ for some threshold value ς. Hence, two nodes can communicate whenever $||x(u) - x(v)|| < (\frac{P_n}{N_o \varsigma})^{1/\beta}$. In particular, we let $P_n = (N_o \varsigma r_n)^\beta$.

[4] The throughput achieved by this scheme is $O(\frac{1}{\sqrt{n \log n}})$ [*bits/second/node*] when n source destination pairs are chosen uniformly at random.

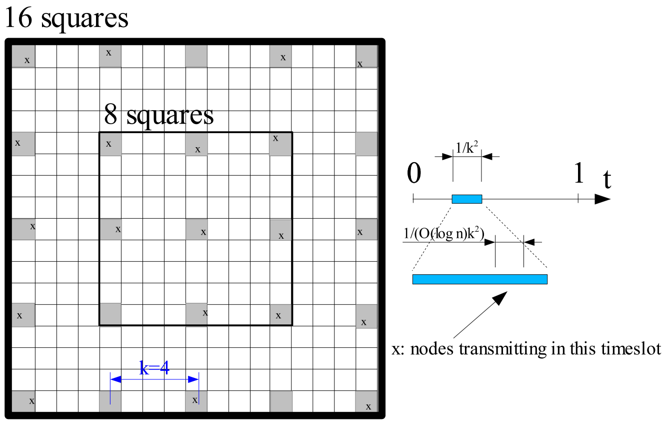


Figure 3.3: Illustration of the TDMA scheduling scheme.

find an upper bound to the interferences, we observe that with this choice, the transmitters in the 8 first closest squares are located at a distance at least 3 (in small squares) from the receiver (see left-hand side of Figure 3.3). This means that the Euclidean distance between the receiver and the 8 closest interferers is at least $2s_n$. The 16 next closest squares are at distance at least 7 (in small squares), and the Euclidean distance between the receiver and the 16 next interferers is therefore at least $6s_n$, and so on. The sum of the interferences I can be bounded as follows:

$$\begin{aligned} I &\leq \textstyle\sum_{i=1}^{\infty} 8iP_n \left[2s_n(2i-1)\right]^{-\beta} \\ &= P_n \left[2s_n\right]^{-\beta} \textstyle\sum_{i=1}^{\infty} 8i \left[(2i-1)\right]^{-\beta} \\ &= (N_o \varsigma r_n)^{\beta} \left[2\tfrac{r_n}{c}\right]^{-\beta} \textstyle\sum_{i=1}^{\infty} 8i \left[(2i-1)\right]^{-\beta} \end{aligned}$$

This term clearly converges if $\beta > 2$. Now we want to bound from below the strength of the signal received from the transmitter. We observe first that the distance between the transmitter and the receiver is at most $\sqrt{2(s_n^2)} \leq 2s_n$. The strength S of the signal at the receiver can thus be bounded by

$$\begin{aligned} S &\geq P_n \min\left\{1, 2s_n^{-\beta}\right\} \\ &= O(1) \end{aligned}$$

Finally, we obtain the following bound on the SINR: $SINR \geq \frac{S}{N_o + I}$. As the above expression does not depend on n, the theorem is proved. ☐

3.2.3 Uniform speed-limited (USL) mobility

Nodes are mobile and move according to the uniform speed-limited (USL) model, a fairly general mobility model defined next. The USL model essentially embodies two conditions: (i) the node distribution at every time step is uniform over the network domain, and (ii) the distance a node can travel over a time step is bounded. We restrict ourselves to the case in which the maximum speed is not dependent on n. In practice, of course, such an assumption is realistic since the maximum speed of the nodes will not increase when new nodes join the network.

Definition 3.5. *A collection of n nodes satisfy the uniform speed-limited (USL) mobility model if the following two conditions are satisfied:*

1. *At every time t, the distribution of nodes over the network domain is uniform.*
2. *For every node u and time t, the distance traveled in the next time step is bounded, i.e., $||x^{(t+1)}(u) - x^{(t)}(u)|| < S$.*

The USL mobility model is quite general. For example, it includes the following cases: (i) The nodes perform independent random walks with bounded one-step displacement. The random walks can be biased, and the displacement distribution does not need to be homogeneous over the node population. We have to assume that the nodes operate in the stationary regime. (ii) The nodes follow the random waypoint model on a torus (RWP). The system has to be in the stationary regime. (iii) The generalized random direction models from [SMS06], which interpolate between the random walk and the random waypoint cases, through a control parameter that can be viewed as the "locality" of the mobility process. (iv) We can also allow for models where nodes do not move independently. As an illustrative example, assume we uniformly place nodes on the square; the nodes then move in lockstep according to any speed-limited mobility process, maintaining their relative positions to each other. Observe that the uniform distribution is maintained for all time steps[5], and that the speed-limited property is true by definition.

We see that the USL class of mobility models is fairly general, and includes many of the models that have been proposed in the literature. For simplicity, we consider that time is discrete. In other words, we look at a snapshot of the network every ΔT seconds. At every time step, the connectivity between nodes will be modified. Hence, we will work with a sequence of connectivity graphs. In order to design a routing algorithm with a low control traffic overhead, we will need to understand how fast the graph distances between nodes can evolve over time. In particular, consider two nodes u and v at distance $d = d^t(u, v)$ at time t. We want to bound the multiplicative factor by which this distance can change in κ time steps. Formally, we define $\kappa(\tau, d)$ as follows:

Definition 3.6. *We say that a communication network is $\kappa(\tau, d)$-smooth if the shortest path distance between any two nodes u an v at shortest path distance d cannot change by more than a factor $\kappa(\tau, d)$ in τ time steps* i.e., $\forall u, v$, *we have:*

$$\max\left\{\frac{d^{(t)}(u,v)}{d^{(t+\tau)}(u,v)}, \frac{d^{(t+\tau)}(u,v)}{d^{(t)}(u,v)}\right\} \leq \kappa(\tau, d)$$

---

[5] note that we move on a torus

Additionally, we simply say that the network is *κ-smooth* if there exists a constant ν such that $\kappa(\nu d, d) \leq \kappa(\nu) = \kappa$ independently of d. In this case, the distances grow at the same speed at all scales. In the sequel, we will bound κ and ν for our model. This USL property holds for a general class of *random trip* mobility models studied in [BV05], where it is shown that the stationary distribution of such mobility models is uniform and ergodic. We restate this theorem without proof.

Theorem 3.2 (Le Boudec and Vojnovic, see [BV05])**.** *The random-trip mobility model has uniform stationary distribution on* $[0, a) \times [0, a)$.

3.2.4 Assumptions

We consider that a time step ΔT is much larger than the round trip time (RTT) through the network *i.e.,* the time scale for mobility is much larger than the time scale for communications. For clarity and in order to simplify the analysis, we will make the assumption that nodes can communicate instantaneously through the network. We also make the assumption that there is a random permutation π on the nodes, and that all nodes in the network know their rank in the permutation. In Section 3.6 we will then drop these assumptions and consider practical aspects of the implementation. Finally, we say that a result holds with high probability (w.h.p.) if it holds with probability at least $(1 - O(\frac{1}{n^\rho}))$, for some constant $\rho > 0$. In Table 3.1, we summarize the notations used in this chapter.

$x^{(t)}(u)$	Position of node u at time t
$d^{(t)}(u, v)$	Shortest path distance from u to v at time t
r_n	Wireless communication radius
$\mathcal{G}(n, r_n)$	Random geometric graph
$\mathcal{B}_R^{(t)}(u)$	Ball of radius R around u
$\kappa(\tau, d)$	$\max\left\{\frac{d^{(t)}(u,v)}{d^{(t+\tau)}(u,v)}, \frac{d^{(t+\tau)}(u,v)}{d^{(t)}(u,v)}\right\} \leq \kappa(\tau, d)$

Table 3.1: Table of notations

3.3 Network Properties

In this section, we prove some properties of the network models presented in Section 3.2, which are necessary to analyze the performance of our algorithm. We focus our attention on the geometric random graph $\mathcal{G}(n, r_n)$, but all the arguments can be extended to the SINR full connectivity model with TDMA scheduling, discussed in Section 3.2.2. In particular, for $\mathcal{G}(n, r_n)$, we now consider the case in which the communication radius r_n is such that $r_n = \sqrt{(1 + \epsilon) \log n} > \log^{1/2} n$, where $\epsilon > 0$.

For uniform speed-limited (USL) mobility models discussed in Section 3.2.3, at each time, the node locations $\{x^{(t)}(u)\}$ have a distribution that is uniform over $[0, \sqrt{n}) \times [0, \sqrt{n})$. Therefore, we now discuss the property of a sequence of geometric random graphs, $\mathcal{G}^{(t)}(n, r_n)$, under USL mobility model. We subdivide the network area on which the nodes live into smaller squares of side $\frac{r_n}{c}$, where c is a constant chosen such that nodes in neighboring squares are connected (see Fig. 3.4) and that an integer number of squares fit into the network area. We arbitrarily set $c = \sqrt{5}$. We number the small squares from 1 to $m = \frac{n}{(r_n/c)^2} = \frac{nc^2}{(1+\epsilon)\log n}$ and denote by E_i the event that small square i does not contain any node, in a sequence of geometric random graphs under USL dynamics over n^ρ time steps, for some constant ρ. In the next theorem, we show that when nodes move according to USL mobility model, all small squares will be populated w.h.p.

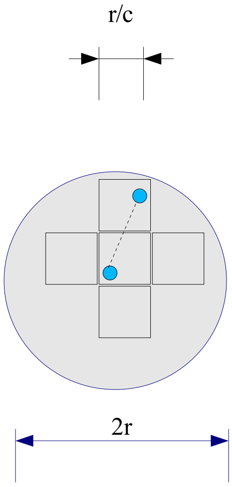


Figure 3.4: Nodes in neighboring squares are connected

Theorem 3.3. *There exists a constant ρ such that if we divide the network into small square of side $\frac{r_n}{c}$ (with $r_n > \sqrt{\log n}$, $c = \sqrt{5}$), at every time step in a sequence of length n^ρ, every small square contains at least one node w.h.p.*

Proof. Consider a sequence of geometric random graphs over $Z = n^\rho$ time steps. Denote by $E_i^{(j)}$ the event the small square i is empty at time j. Let

$m = \frac{n}{(r_n/c)^2}$. We can compute:

$$\begin{aligned}
\mathcal{P}\left[\bigcup_{j=1}^{Z}\bigcup_{i=1}^{m} E_i^{(j)}\right] &\leq Z\sum_{i=1}^{m}\mathcal{P}\left[E_i^{(j)}\right] \\
&\overset{(a)}{=} Z\sum_{i=1}^{m}(1-\tfrac{1}{m})^n \\
&\overset{(b)}{\leq} Z\sum_{i=1}^{m} e^{-\frac{n}{m}} \\
&= Zme^{-\frac{n}{m}} \\
&= Z\frac{nc^2}{(1+\epsilon)\log n}e^{-\frac{nc^2(1+\epsilon)\log n}{n}} \\
&\leq Z\frac{nc^2}{(1+\epsilon)\log n}\frac{1}{n^{(1+\epsilon)c^2}} \\
&\overset{(c)}{\leq} Z\frac{c^2}{(1+\epsilon)n^{\epsilon c^2}\log n} \\
&\leq O(\tfrac{n^\rho}{n^{\epsilon c^2}}) \\
&= O(\tfrac{1}{n^{\epsilon c^2-\rho}})
\end{aligned}$$

where (a) uses the USL model which implies, due to uniform (marginal) distribution of nodes $\mathcal{P}\left[E_1^{(j)}\right] = (1-\frac{1}{m})^n$, (b) follows because $(1-\frac{1}{m})^n \leq e^{-\frac{n}{m}}$ (see [MR95], page 434), and (c) follows because $c = \sqrt{5} > 1$. We can now choose ρ such that $\epsilon c^2 - \rho > 0$ and the result follows. ☐

It is immediate that in a single instantiation of the connectivity graph (*i.e.,* at a given time slot), every small square is populated w.h.p.

Corollary 3.1. *With probability at least* $(1 - O(\frac{1}{n^\epsilon}))$*, there is no empty small square in a sequence of length* 1*.*

We are now ready to show that at every time step in a sequence of n^ρ connectivity graphs, the connectivity graph is doubling w.h.p. Since we have a USL mobility model, any graph $\mathcal{G}^{(t)}(n, r_n)$ is statistically identical to $\mathcal{G}(n, r_n)$.

Theorem 3.4. $\mathcal{G}(n, \sqrt{(1+\epsilon)\log n})$ *are doubling w.h.p.*

Proof. By Corollary 3.1, all small squares of side $\frac{r_n}{c}$ contain at least one node w.h.p. Consequently, neighboring squares (vertically and horizontally) have at least one communication link. Denote by $\mathcal{L}_n$ the grid having the small squares as vertices, and with edges between vertical and horizontal neighbors. Consider a ball $B_{upper} = \mathcal{B}_{2R}^{\mathcal{G}}(u)$ (*i.e.,* a ball of radius $2R$ in the communication graph $\mathcal{G}(n, r_n)$) centered around some node u. Clearly, all nodes in B_{upper} must be contained in a square which is part of $\mathcal{B}_{4Rc}^{\mathcal{L}_n}(u)$ *i.e.,* $B_{upper} \subseteq \mathcal{B}_{4Rc}^{\mathcal{L}_n}(u)$. This follows from the fact that no node in B_{upper} can be further away from u than $2Rr_n$ in Euclidean distance, that the grid $\mathcal{L}_n$ is fully connected w.h.p., and that one link in the grid has length $\frac{r_n}{c}$. Similarly, one can see that $\mathcal{B}_{R}^{\mathcal{L}_n}(u) \subseteq B_{lower} = \mathcal{B}_{R}^{\mathcal{G}}(u)$. This is a consequence of the fact that $\mathcal{L}_n$ is a subgraph of $\mathcal{G}$, *i.e.,* two nodes in small squares R hops a part in $\mathcal{L}_n$ cannot be more than

R hops apart in $\mathcal{G}$ (see Fig. 3.5). For an appropriately chosen constant α, we have:

$$B_{upper} \subseteq \mathcal{B}^{\mathcal{L}_n}_{4Rc}(u) \subseteq \bigcup_{j=1}^{\alpha} \mathcal{B}^{\mathcal{L}_n}_{R}(v_j) \subseteq \bigcup_{j=1}^{\alpha} \mathcal{B}^{\mathcal{G}}_{R}(v_j) \tag{3.2}$$

and $\mathcal{G}(n, \sqrt{(1+\epsilon)\log n})$ is *doubling* (*i.e.,* any ball of radius $2R$ can be covered by a constant number of balls of radius R). ☐

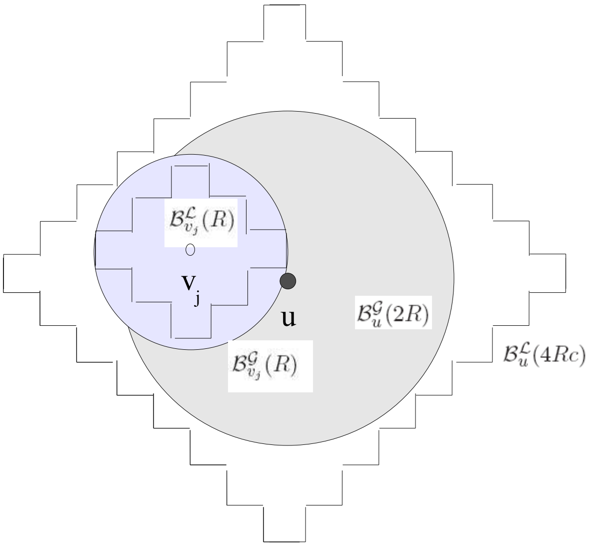

$\mathcal{B}^{\mathcal{L}}_{v_j}(R)$
v_j
u
$\mathcal{B}^{\mathcal{G}}_{u}(2R)$
$\mathcal{B}^{\mathcal{L}}_{u}(4Rc)$
$\mathcal{B}^{\mathcal{G}}_{v_j}(R)$

Figure 3.5: Any ball of radius $2R$ can be covered by a constant number of balls of radius R. In order to show this result, one can work with balls in the underlying lattice.

Note that the close relationship of the geometric random graph and the corresponding grid (lattice) is the basic insight behind this result. Additionally, note that it is possible to build a deterministic geometric graph for which this property does not hold. We present such a construction in Appendix A-1. Further, one can show that $\mathcal{G}(n, r_n)$ are *not* doubling w.h.p when $r_n < \sqrt{\log n}$. We prove this result in Appendix A-2. At this point, we would like to emphasize that even though we analyze networks in which the nodes are uniformly distributed on a square area, the doubling property is a much more powerful tool. Indeed, our results and algorithms depend only on the doubling constant. Consequently, the algorithms and the bounds can be applied to any other type of networks or node configuration which lead to a doubling connectivity graph. For instance, one can consider the network shown in Figure 3.2, described in Section 3.2.1. It can easily be shown by using a technique similar to the one

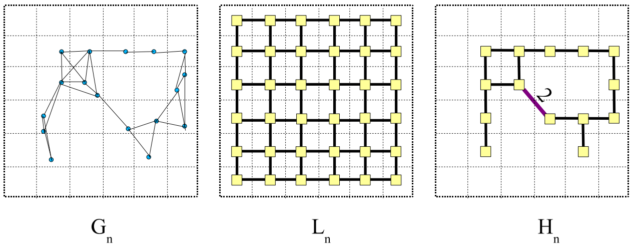


Figure 3.6: Graphs $\mathcal{G}_n$, $\mathcal{L}_n$ and $\mathcal{H}_n$. The network area is divided into squarelets of side $\frac{r_n}{c}$ such that nodes in horizontally and vertically adjacent squarelets are guaranteed to be within communication range.

used in Theorem 3.4 that this network is doubling. While we can seamlessly apply our routing algorithm to such a network, any classical geographic routing algorithm would fail or require a high control traffic overhead to get out of dead-ends. This is because packets would get stuck against the wall when routed from the lower to the upper part of the network. In turn, this would considerably degrade the performance in terms of stretch and control traffic overhead with respect to the same network without a wall. In the next subsection we prove a set of sufficient conditions for a wireless networks to have a constant doubling dimension.

3.3.1 Inhomogeneous Topologies

In the first part of this subsection, we show that under certain conditions, the presence of topological holes (obstacles) in the network does not change the doubling property. In particular, we are interested in how we can alter the topology of a fully connected random geometric graph by removing nodes while still preserving the doubling property. In the second part, we will generalize this idea to arbitrary metric spaces. Consider a $\mathcal{G}(n, r_n)$ with $r_n > \sqrt{\log n}$, such that full connectivity is guaranteed. The network area is divided into squarelets of side $\frac{r_n}{c}$, where c is chosen such that nodes in horizontally and vertically adjacent squarelets are guaranteed to be within communication range. We now arbitrarily select squarelets and remove all nodes they contain. We denote the new graph we obtain by $\mathcal{G}_n$. We denote by $\mathcal{L}_n$ the full grid where the squarelets are vertices and by $\mathcal{H}_n$ the corresponding grid in $\mathcal{G}_n$ *i.e.,* the thinned out grid obtained by selecting only non-empty squarelets in $\mathcal{G}_n$. In $\mathcal{L}_n$, we add an edge between horizontally and vertically adjacent squarelets (see Fig. 3.6). In $\mathcal{H}_n$, we first add a an edge between horizontally and vertically adjacent squarelets containing at least one node. Then, for every pair of squarelets containing nodes that can communicate directly, we add an edge of weight corresponding to the distance between those two squarelets in $\mathcal{L}_n$. We add the edges from the shortest to the longest one, and only if no path of the same length already exists in $\mathcal{H}_n$. We can now define a topological hole as follows:

Definition 3.7 (Topological Hole)**.** *A set of horizontally, vertically and horizontally adjacent empty squarelets in the graph $\mathcal{H}_n$ is called a* hole *if adding a (virtual) vertex in all of the squarlets in that set modifies the distance between at least two vertices in $\mathcal{H}_n$.*

Let us denote by $\mathcal{V}_k$ the k^{th} hole ($k = 1, 2, 3, ...$). We define the perimeter $p(\mathcal{V}_k)$ of $\mathcal{V}_k$ as 2 times the maximum distance between any two vertices on the border of the hole *i.e,* in squarelets adjacent to the empty squarelets defining the hole. Note that for all u, v, we have $d^{\mathcal{H}_n}(s_u, s_v) \geq d^{\mathcal{G}_n}(u, v)$.

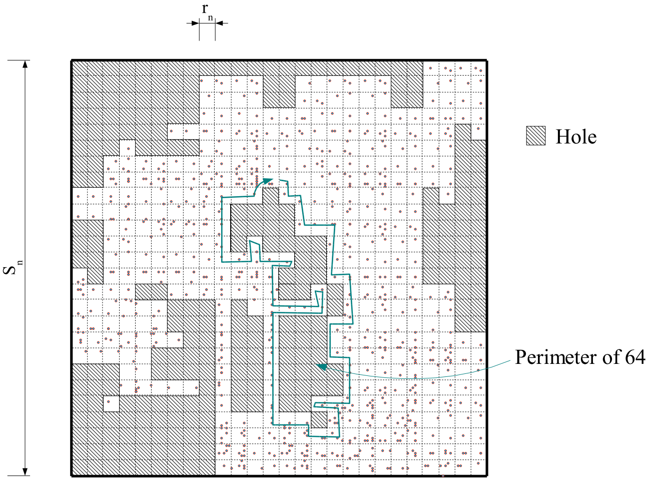


Figure 3.7: A network with holes

Theorem 3.5. *Let $P = \max_k p(\mathcal{V}_k)$. Then, the doubling dimension is upper bounded by $O(P^2)$.*

Proof. Consider a ball $\mathcal{B}^{\mathcal{G}_n}_{2R}(u)$ centered at u in $\mathcal{G}_n$. First, observe that $\mathcal{B}^{\mathcal{G}_n}_{2R}(u) \subseteq Box^{\mathcal{L}_n}_{2Rc}(u)$, where $Box^{\mathcal{L}_n}_{2Rc}(u)$ is the box centered at the squarelet containing u in $\mathcal{L}_n$ which contains all nodes at "maximum norm" $2Rc$ (*i.e.,* l_∞-norm) from this squarelet. In other words, all nodes within $2R$ hops from u in $\mathcal{G}_n$ must be in a squarelet contained in this box. We will now cover this box with smaller boxes $Box^{\mathcal{L}_n}_{\max\{1,\lceil\frac{R}{4\gamma}\rceil\}}(s_{v_i})$. We need $\left\lceil \frac{16R^2c^24\gamma^2}{R^2} \right\rceil = 64c^2\gamma^2$ such boxes at most. Consider the same small boxes in $\mathcal{H}_n$. Pick one non-empty squarelet s_{v_i} in each such small boxes. Note that the maximum hop distances between two squarelets in such a small box in $\mathcal{L}_n$ is at most $\frac{R}{\gamma}$. For each of these hops, we might have to make a detour of at most P steps. Consequently, the same two squarelets could be at distance at most $\frac{PR}{\gamma}$ in $\mathcal{H}_n$. Observe now that for any two nodes v and w contained in squarelets s_v and s_w respectively, we have $d^{\mathcal{H}_n}(s_v, s_w) \geq d^{\mathcal{G}_n}(u, w)$. For each squarelet s_{v_i}, we pick one node v_i contained in this squarelet. Hence, for all nodes w contained in this small box, we have $d^{\mathcal{G}_n}(v_i, w) \leq d^{\mathcal{H}_n}(s_{v_i}, s_w) \leq \frac{PR}{\gamma}$. By setting $\gamma = P$, we obtain the claim. The proof is illustrated in Figure 3.8. ☐

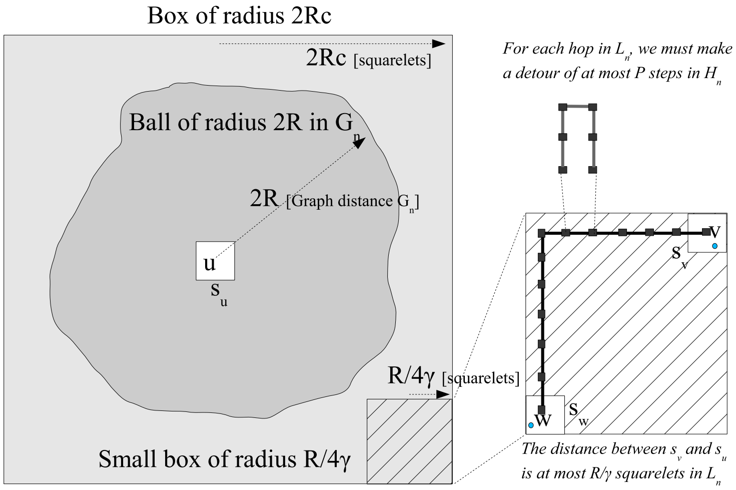

Box of radius 2Rc
2Rc [squarelets]
Ball of radius 2R in G_n
2R [Graph distance G_n]
u
s_u
R/4γ [squarelets]
Small box of radius R/4γ
For each hop in L_n, we must make a detour of at most P steps in H_n
v
s_v
w
s_w
The distance between s_v and s_u is at most R/γ squarelets in L_n

Figure 3.8: All nodes in a ball of radius $2R$ in $\mathcal{G}_n$ are contained in a box of radius $2Rc$ squarelets in $\mathcal{L}_n$. This large box in $\mathcal{L}_n$ can be covered by $O(\gamma^2)$ smaller boxes of radius $R/(4\gamma)$ squarelets. Any two nodes (say v and w) inside the same small box must be in squarelets at distance at most R/γ in $\mathcal{L}_n$. By hypothesis, the perimeter of any hole is at most P. Consequently, the distance in $\mathcal{H}_n$ between v and w is at most R squarelets. All nodes within R squarelets are also within distance R in $\mathcal{G}_n$, as the graph distance in $\mathcal{H}_n$ is always larger or equal to the graph distance in $\mathcal{G}_n$. Hence, if we select one node in each of the small boxes, we are guaranteed to cover all nodes in the ball of radius $2R$.

We can extend this result to the case where the network can be divided into convex sets. We define a convex set in $\mathcal{G}_n$ with slack as follows:

Definition 3.8. *Let Ψ be a set of nodes in $\mathcal{G}_n$. Let $\mathcal{H}_n{}^{(\Psi)}$ be the squarelets in $\mathcal{H}_n$ containing at least one node in Ψ. We say that the set Ψ is* convex *if $\forall u, v \in \Psi$, $d^{\mathcal{H}_n{}^{(\Psi)}}(s_u, s_v) = d^{\mathcal{L}_n}(s_u, s_v)$, where s_u and s_v are the squarelets containing u and v* i.e., *there must be at least one shortest path inside the convex set. We say that the set Ψ is* convex with slack *ϖ if $d^{\mathcal{H}_n{}^{(\Psi)}}(s_u, s_v) \leq \varpi d^{\mathcal{L}_n}(s_u, s_v)$.*

The perimeter of a convex set is defined the same way as the perimeter of a holes *i.e.,* as two times the largest distance in the set. We can now state the following theorem

Theorem 3.6. *Let* $\zeta_1, \zeta_2, ..., \zeta_q$ *be a partition of the network into* q *convex sets with slack* $\varpi_1, \varpi_2, ..., \varpi_q$ *respectively. Let* $\vartheta_1, \vartheta_2, ..., \vartheta_q$ *denote the perimeter of the convex sets. The doubling dimension is then upper bounded by* $\max_{u,\rho} 64c^2 \sum_{\zeta_i : \zeta_i \cap Box_{\rho}^{\mathcal{L}_n}(s_u) \neq \emptyset} (\left\lceil \frac{\vartheta_i}{\frac{\rho}{\varpi_i}} \right\rceil)^2$.

Proof. In the proof of Theorem 3.5, we have shown that any ball of radius $2R$ around some node u is contained in a box $Box_{\rho}^{\mathcal{L}_n}(s_u)$, where $\rho = 2Rc$ (implying $R = \frac{\rho}{2c}$). We can cover each convex set ζ_i intersecting this box with at most $16\frac{\vartheta^2}{\frac{R^2}{\varpi_i^2}}$ small boxes of side $R/4\varpi_i$, as shown in Theorem 3.5. Indeed, a convex area of perimeter ϑ can always be included in a square area of side ϑ. If a convex set has slack ϖ_i, and is covered by small boxes of side $R/4\varpi_i$, then by selecting one node in each of the small boxes, we guarantee that all nodes in the convex set are within R hops in $\mathcal{G}_n$ of one of the aforementioned selected nodes. If the box $Box_{\rho}^{\mathcal{L}_n}(s_u)$ is partitioned into several convex sets, selecting $64c^2(\left\lceil \frac{\vartheta_i}{\frac{\rho}{\varpi_i}} \right\rceil)^2$ nodes in each convex set ζ_i intersecting this box will in turn guarantee that all nodes in the box are covered. Hence, the total number of balls of radius R needed to cover any ball of radius $2R$ is upper bounded by the maximum over all boxes (all centers and radii), of the sum of the number of small boxes required to cover all the convex sets that intersect it. □

In practice, this result implies that if we are given a decomposition of the network into convex sets, we can bound the overall doubling dimension given the doubling dimension of each set separately. Further, this result implies that networks that consist of a small number of convex areas, which can each contain arbitrarily many small holes, have a low complexity in terms of doubling dimension. We will now relate the "shapes" of a topological hole to the doubling dimension. In particular, we will show that one can relate the doubling dimension to the maximum number of connected components in any square subarea.

Theorem 3.7. *For any* $\gamma \geq 2$*, the doubling dimension* α *is such that*

$$\alpha \leq 4\gamma^2 c^2 \max_{Box_{R/\gamma}^{\mathcal{L}_n}(u)} \left\{ \#\ \textit{of convex components with slack } \gamma \textit{ in } Box_{R/\gamma}^{\mathcal{L}_n}(u) \right\}$$

Proof. In the proof of Theorem 3.5, we have shown that any ball of radius $2R$ around some node u is contained in a box $Box_{\rho}^{\mathcal{L}_n}(s_u)$, where $\rho = 2Rc$. In turn, we showed that by dividing this box into smaller boxes of side R/γ, and by selecting one node in each box, we could cover the larger ball of radius

$2R$. Now, in each small box of side R/γ, the presence of holes might create several disconnected components. However, we know that inside each such component, we can cover any convex subset with slack γ with one nodes. The result follows. ☐

This last result gives us a characterization of the alterations we can make to a fully connected $\mathcal{G}(n, r_n)$ network, while only affecting the doubling dimension by a constant factor. In particular, we can remove nodes as long as we do not create too many convex and disconnected components in any square subarea. Note that we can still remove arbitrarily many nodes as long as we only create small holes. Theorems 3.5 imply that topologies such as the one shown in Fig. 3.9 have a constant doubling dimension. The results stated above are special cases of the more general result detailed in the sequel. Indeed, we can relate the doubling dimension in a metric space to the doubling dimension in another

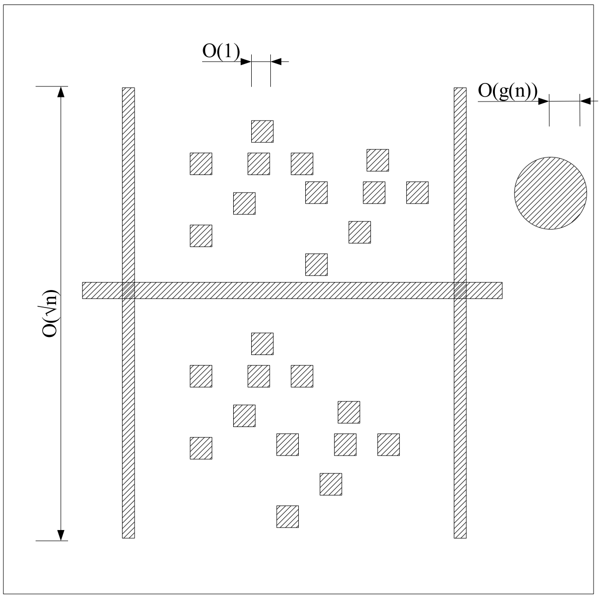


Figure 3.9: A network with topological holes and a constant doubling dimension. The size of the large holes grow with n, but the network can be divided into a constant number of areas, each being convex with slack $O(1)$ *i.e.,* each of the convex areas contains only obstacles with a constant perimeter or that can only increase the distance between nodes by a constant factor. Note that even though the doubling dimension is low, greedy geographic forwarding of packets would fail as packets would get stuck in dead-ends against the holes. Hatched regions are holes.

metric space if we know the distortion of the embedding that maps the points in one metric space to the points in the other metric space. The example above is a special case of that setup where we map the nodes of a graph to points in Euclidean space. Consider two metric spaces (X, d) and (X', d'), where d and d' are distance functions which define a metric on the sets of point X and X'. We could for instance consider the two metric spaces $(\mathcal{X}, ||.||)$ and $(\mathcal{H}, d(.,.))$ *i.e.,* the points in the plane with the Euclidean distance and the nodes in the graph with the shortest path distance. A *metric embedding* is a bijective function $\phi : \mathcal{X} \rightarrow \mathcal{X}'$ which associates to a point in one metric space a point in another metric space. We have already defined the notion of embedding in Section 2.2, but we repeat it here for the convenience of the reader.

Definition 3.9 (Distortion of an Embedding)**.** *A mapping $\phi : X \rightarrow X'$ where (X, d) and (X, d') are metric spaces, is said to have distortion at most D, or to be a* D-embedding*, where $D \geq 1$, if there is a $K \in (0, \infty)$ such that $\forall x, y \in X$,*

$$Kd(x, y) \leq d'(\phi(x), \phi(y)) \leq KDd(x, y)$$

if X' is a normed space, we typically require $K = 1$ or $K = \frac{1}{D}$. An embedding has distortion D with slack ϵ if all but an ϵ fraction of node pairs have distortion D under ϕ. Additionally, one can loosen this definition by allowing slack. The slack is said to be uniform *if each node has distortion at most D to a $1 - \epsilon$ fraction of the other nodes. Finally, an embedding with distortion D and slack ϵ is* coarse *if for every node u the distortion is bounded to a node a distance greater than $r_\epsilon = inf\left\{rs.t\ |\mathcal{B}^X_r(u)| > \epsilon n\right\}$.*

The doubling dimension of a metric space embedded into another metric space can be bounded as follows:

Theorem 3.8 (Bounding the Doubling Dimension)**.** *Consider a metric space $(\mathcal{H}, d)$ embedded in another metric space $(\mathcal{E}, d')$ by a function ϕ. Let the doubling dimension of $\mathcal{E}$ be β. Let the distortion of this embedding be D. Then, $\mathcal{H}$ has doubling dimension α with $\alpha \leq O((2D)^{\log \beta})$.*

Proof. Choose any node $u \in \mathcal{H}$. If the above condition is fulfilled, the images of all nodes in $\mathcal{B}^{\mathcal{H}}_{2R}(u)$ can be at distance d' at most $2KDR$ from u at $\phi(u)$. Hence, $\phi(\mathcal{B}^{\mathcal{H}}_{2R}(u)) \subset \mathcal{B}^{\mathcal{E}}_{2KDR}(\phi(u))$. We will now try to cover $\phi(\mathcal{B}^{\mathcal{H}}_{2R}(u))$ by as few balls $\mathcal{B}^{\mathcal{E}}_{RK}(\phi(v))$ as possible (see Fig. 3.10, which illustrates this setup in the case when $\mathcal{H}$ is a graph and $\mathcal{E}$ the Euclidean space). To do so, let us cover $\mathcal{B}^{\mathcal{E}}_{2KDR}(\phi(u))$ by small balls of radius KR in $\mathcal{E}$. Covering $\mathcal{B}^{\mathcal{E}}_{2KDR}(\phi(u))$ will require at most $\beta^{\log 2D}$ balls of radius KR in $\mathcal{E}$, given that $\mathcal{E}$ has doubling dimension β. We know that $d(u, v) \leq d'(u, v)/K$, by definition 3.9. Consequently, $\phi^{-1}(\mathcal{B}^{\mathcal{E}}_{RK}(\phi(v))) \subset \mathcal{B}^{\mathcal{H}}_{R}(v)$. We can conclude that $\mathcal{B}^{\mathcal{H}}_{2R}(u) \subset \bigcup_{j=1}^{\beta^{\log 2D}} \mathcal{B}^{\mathcal{H}}_{R}(v_j)$. ☐

The presence of large obstacles in the network does not necessarily imply that the network is not doubling. In particular,

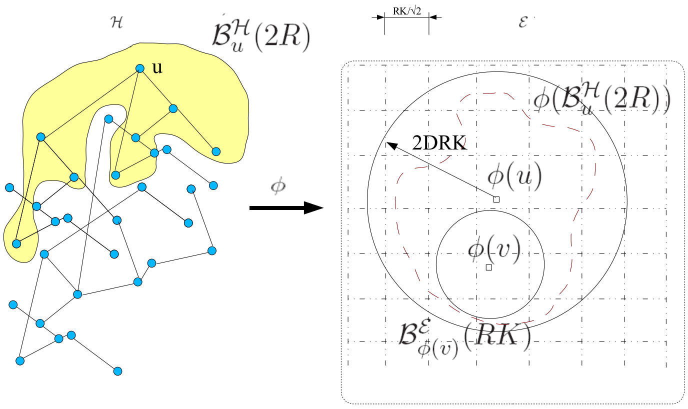

$\mathcal{H}$
$\mathcal{B}_u^{\mathcal{H}}(2R)$
u
ϕ
RK/√2
$\mathcal{E}$
$\phi(\mathcal{B}_u^{\mathcal{H}}(2R))$
2DRK
$\phi(u)$
$\phi(v)$
$\mathcal{B}^{\mathcal{E}}_{\phi(v)}(RK)$

Figure 3.10: Effect of embedding on doubling dimension

Theorem 3.9. *Consider a metric space $\mathcal{E}$ with doubling dimension β. A metric space $\mathcal{H}$ that can be divided in k sets $S_1, S_2, ..., S_k$, such that each set embeds individually with distortion D_i into $\mathcal{E}$ has doubling dimension at most $\sum_{j=1}^{k} \beta^{2\log 2D_j}$.*

Proof. Consider any ball of radius $2R$ in $\mathcal{H}$, such that the nodes in the ball belong to at least two different sets (otherwise the theorem is clearly true). Note that the radius of each of these subsets can be at most $4R$. Consequently, we now that the part of the ball that belongs to S_i can be covered by at most $\beta^{2\log 2D_i}$ (by applying Theorem 3.8 to cover a ball of radius $4R$ by balls of radius R). The theorem follows. ☐

We can now broaden the class of communication networks that have low doubling dimension. In particular, if we can subdivide the communication graph into a constant number of subsets, such that each one embeds with constant distortion into the Euclidean plane, the whole network is doubling. Consequently, topologies such as the one shown in Figure 3.11 are doubling. In this example, we embed an unweighted graph into the Euclidean plane. Note that the minimal Euclidean distance between nodes should be ρr_n (for some constant ρ), such that $\rho r_n d(u,v) \leq ||x(u) - x(v)|| \leq O(1)\rho r_n$. If this equation is true for all pairs of nodes, then the distortion is $O(1)$. There is an issue when the nodes are neighbors in the communication graph, as the above rule implies that the Euclidean distance between such pairs of nodes should then be at least $O(r_n)$. However, we can ignore the distances below 2 as we will not cover balls of radius 1 (since we have a broadcast medium, the degree of a node does not impact the communication overhead). In such cases, it is obvious that geographic routing would fail, even though the inherent complexity of the network is low. Indeed, packets would get stuck against walls. Remarkably, our routing algorithm is oblivious to the topology and only depends on the doubling dimension. Hence, there is absolutely no need to detect or identify obstacles. The communication overhead will simply depend on the doubling dimension.

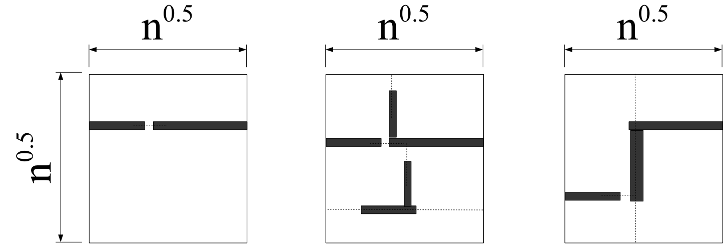


Figure 3.11: A set of doubling network topologies. The network is dense, and made inhomogeneous by the walls, which do no allow transmissions to go through. Note that the walls stretch when n grows, such that the network wide distortion also grows with n. Dashed lines indicate the separation into sets.

3.3.2 Sequences of Communication Graphs

In this subsection we study the behavior of a sequence of communication graphs, *without* any obstacles. We show that a sequence of $\mathcal{G}^{(t)}(n, r_n)$ of length n^ρ, for some constant ρ, with the USL mobility model is κ-smooth. As already seen in Theorem 3.4, such a sequence of graphs is doubling at every time instant.

Theorem 3.10. *A sequence of $\mathcal{G}^{(t)}(n, r_n)$ of length $\leq n^\rho$, where nodes move according to the USL mobility model with maximum constant speed S is*

$$max \left\{ \frac{r_n d^{(t)}}{\frac{r_n d^{(t)}}{\sqrt{5}\sqrt{2}} - 2\sqrt{5}\sqrt{2}\tau S}, \sqrt{5}\sqrt{2}(1 + \frac{2\tau S\sqrt{5}\sqrt{2}}{r_n d^{(t)}}) \right\}$$

smooth w.h.p.

Proof. Consider two nodes u and v at Euclidean distance $q_2^{(t)} = ||x_u - x_v||_2$ at time t. Let $q_1^{(t)} = ||x_u - x_v||_1 = \sum_{m=1}^{2} |x_m(u) - x_m(v)|$. Further, denote by $d^{(t)} = d^{(t)}(u, v)$ their shortest path distance at time t. One can see that $\frac{q_2^{(t)}}{r_n} \leq d^{(t)} \leq \frac{\sqrt{5}\sqrt{2}q_2^{(t)}}{r_n}$. Indeed, the shortest possible path will follow a straight line between u and v. The length of this line is $q_2^{(t)}$ and one hop can be of length at most r_n. In the worst case, the shortest path from u to v will follow the shortest path in the grid formed by the small squares of side $\frac{r_n}{c} = \frac{r_n}{\sqrt{5}}$, which exists w.h.p. Recall that we can only guarantee horizontal and vertical connectivity between small squares. The number of small squares in this path will be at most $\frac{\sqrt{5}q_1^{(t)}}{r_n}$. One can easily show that $q_1^{(t)} \leq \sqrt{2}q_2^{(t)}$. Let $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ sx_1 \end{pmatrix} = (x_u - x_v)$. We have

$$\begin{aligned} q_2^{(t)} &= \sqrt{x_1^2 + x_2^2} = x_1\sqrt{1+s^2} \\ &= (1+s)x_1\tfrac{\sqrt{1+s^2}}{1+s} = q_1^{(t)}\tfrac{\sqrt{1+s^2}}{1+s}. \end{aligned}$$

Since, we have $\frac{q_2^{(t)}}{q_1^{(t)}} = \frac{\sqrt{1+s^2}}{1+s}$, the term is maximized when $s = 1$. In Figure 3.12, we illustrate this point. Similarly, at time $t+\tau$, the shortest path distance

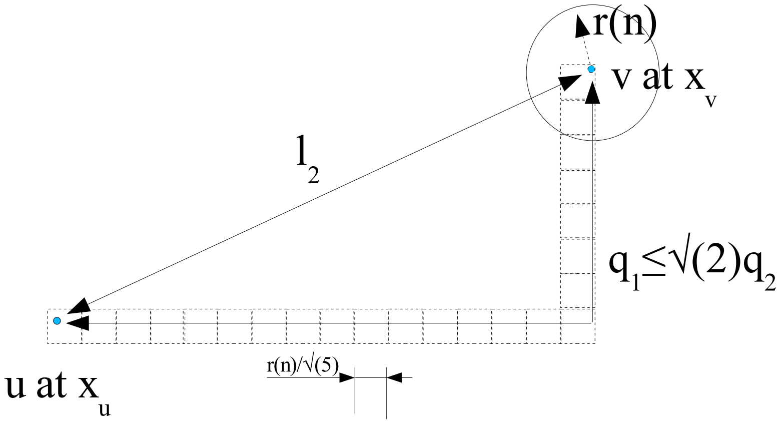


Figure 3.12: Upper and lower bounds for the shortest path

will be bounded by $\frac{q_2^{(t+\tau)}}{r_n} \leq d^{(t+\tau)} \leq \frac{\sqrt{5}\sqrt{2}q_2^{(t+\tau)}}{r_n}$. However, we know that the Euclidean distance can change by at most $2\tau S$ in τ time steps[6]. Consequently,

$$\frac{q_2^{(t)} - 2\tau S}{r_n} \leq d^{(t+\tau)} \leq \frac{\sqrt{5}\sqrt{2}(q_2^{(t)} + 2\tau S)}{r_n} \tag{3.3}$$

We can now bound the multiplicative stretch as follows: Hence,

$$\begin{aligned} & max\left\{(\tfrac{1}{\sqrt{5}\sqrt{2}} - \tfrac{2\tau S}{\sqrt{5}\sqrt{2}q_2^{(t)}})^{-1}, \sqrt{5}\sqrt{2}(1 + \tfrac{2\tau S)}{q_2^{(t)}})\right\} \\ &= max\left\{\sqrt{5}\sqrt{2}\tfrac{q_2^{(t)}}{q_2^{(t)}-2\tau S}, \sqrt{5}\sqrt{2}(1 + \tfrac{2\tau S}{q_2^{(t)}})\right\} \\ &= max\left\{\frac{r_n d^{(t)}}{\frac{r_n d^{(t)}}{\sqrt{5}\sqrt{2}}-2\sqrt{5}\sqrt{2}\tau S}, \sqrt{5}\sqrt{2}(1 + \tfrac{2\tau S\sqrt{5}\sqrt{2}}{r_n d^{(t)}})\right\} = \kappa(\tau, d) \end{aligned}$$

---

[6] One can show that this remains true even if the nodes are reflected on the borders of the network

□

One can now observe that the time it takes to multiply the shortest path distance between two nodes at distance d is proportional to d. Note that the larger the communication radius r_n, the smaller κ. Hence, the distance grows at most linearly with time. In particular, we have:

Corollary 3.2. *There exist constants ν and κ defined in the proof such that a sequence of n^ρ connectivity graphs, under the USL mobility model with maximum constant speed S, is κ-smooth w.h.p.*

Proof. By theorem 3.10, we know that the sequence is

$$max\left\{\frac{r_n d^{(t)}}{\frac{r_n d^{(t)}}{\sqrt{5}\sqrt{2}} - 2\sqrt{5}\sqrt{2}\tau S}, \sqrt{5}\sqrt{2}(1+\frac{2\tau S\sqrt{5}\sqrt{2}}{r_n d^{(t)}})\right\}$$

-smooth w.h.p. Note that both terms decrease as a function of the communication radius r_n. Hence, we can set $r_n = 1$ without decreasing $\kappa(\tau, d)$. Similarly, both terms go down when the distance $d^{(t)}$ goes up. We can therefore also set $d^{(t)} = 1$, which is the smallest possible distance in an unweighted graph. Consequently, if we set $\tau = \nu d^{(t)} = \nu$, we can now write

$$\kappa(\tau, d) \leq max\left\{\frac{1}{\frac{1}{\sqrt{5}\sqrt{2}} - 2\sqrt{5}\sqrt{2}\nu S}, \sqrt{5}\sqrt{2}(1+2\nu S\sqrt{5}\sqrt{2})\right\}$$

which is constant for ν constant. □

3.4 Routing Algorithm

We develop the routing algorithm and its performance analysis for a general class of dynamic networks which produce a sequence of doubling and smooth connectivity graphs. We have seen in Sections 3.2 and 3.3 that this applies to a class of wireless connectivity models with USL mobility. For notational convenience we illustrate the ideas for a sequence $\mathcal{G}^{(t)}(n, r_n)$ geometric random graphs with USL mobility.

Node identifier	*distance [hops]*	*level*	*next hop*
⋮	⋮	⋮	⋮

Table 3.2: Routing Table RT

We decompose a time step into two phases: a *beaconing* phase and a *forwarding* phase. In the former phase, a set of routes are established by letting all or a subset of nodes flood the network at geometrically decreasing radii and nodes register with beacon nodes. In the latter phase, this subset of routes is then utilized by source nodes to efficiently search for the destination. Every node is equipped with a routing table as shown in Table 3.2. We first describe two procedures used in the beaconing and the routing phase.

flood(R,level) procedure :

When a node u initiates the $flood(R, level)$ procedure, it broadcasts a *flood packet* as shown in Table 3.3 to its direct neighbors in $\mathcal{G}(n, r_n)$. The *hop count* field is initialized to 0 and the content of the *Level* field is set to the level of the beacon. How the level of the beacon is determined will be specified in the sequel. All nodes can compute the maximum hop count given the value of the level field in the packet. The neighbors which receive this packet, after increasing the hop count by 1, add an entry to their routing table for node u if no entry for the same node and level with lower or equal hop count is present in the RT. The *next hop* field is set to the identifier of the node from which the packet was received. The level field in the routing table is set to the level given in the packet. In turn, the nodes which got the packet from u broadcast this packet to their neighbors. The latter follow the same procedure and update the routing table if necessary. The packet is discarded when the hop count reaches the maximum hop count (which is a function of the level). Note that with this procedure, every node forwards the packet at most once and the distance added to the routing table is the shortest path distance in $\mathcal{G}(n, r_n)$. This procedure also allows us to establish a reverse path from all nodes that get the packet back to u. Indeed, it suffices for all these nodes to store the identifier of the node from which they received the packet[7]. Further, for any node v, that reverse path is a shortest path to u.

Pkt. Type	*Node Id.*	*Hop Count*	*Level*
$O(1)$ bits	$O(\log n)$ bits	$O(\log n)$ bits	$O(\log \Delta)$ bits

Table 3.3: Flood Packet

probe(relay,destination) procedure :

This procedure consists in sending a *probe packet* (see Table 3.4) to a "relay" node for which the source has an entry in its routing table. The relay node will set the success bit to 1 if it has an entry for the destination and 0 otherwise. We will make sure that all nodes on the path between the source and the relay node have an entry for the relay node in their routing table. Additionally, nodes on the path add a temporary entry for the source in the routing table. They set the *next hop* field to the identifier of the node from which they received the packet and leave the *level* and *distance* field empty. Upon receiving the packet, the relay node can either answer to the source on the reverse path we just created if the answer is negative. Alternatively, it can take action as explained in the sequel if it has an entry for the destination.

---

[7] The packet for which they modified their routing table

Pkt. Type	*Relay Id.*	*Dest. Id.*	Success
$O(1)$ bits	$O(\log n)$ bits	$O(\log n)$ bits	1 bit

Table 3.4: Probe Packet

We now separately detail the beaconing and the routing algorithms underlying our routing protocol

3.4.1 Beaconing Algorithm

In this subsection, we start by describing the first time step, when nodes have not yet moved and no information has been exchanged (the full algorithm is described in Algorithm 2). In a static network, the information exchanged in this first step would be sufficient to setup a complete routing infrastructure. On the other hand, in a mobile environment, we would need to cope with the dynamic topology and constantly update the routing tables. How we deal with a dynamic environment is explained at the end of the subsection. Let the *cover radius* at level i, for $i = 1, ..., \log \Delta$ (Δ being the diameter of the network), be defined as $r_i = 2^i$ and the *flooding radius* at level i be defined as $f_i = \kappa(r_{i+1} + r_i)$, where κ is a constant chosen such that $\kappa(\nu d, d) \leq \kappa d$, $\forall d$. In order for the routing algorithm to work properly, it is crucial that beacons at level i are within the flooding radius of the beacons at level $i + 1$, if they have common nodes inside their cover radii (see Fig. 3.13). This is why we define the flooding radius above. Note that if the network is static, we can set κ to 1. Else, the value of κ depends on how mobile the nodes are (see Def. 3.6).

The idea of the algorithm is to build a hierarchical cover of the network *i.e.,* we would like every node in the network to be within r_i hops of a beacon node at every level i. We say that when a node is within r_i of a beacon b at level i, it is covered and a member of b's *cluster* at level i. It can only be in one cluster at every level. To achieve this, we let the nodes flood in a random order which can change at every time step. Before explaining how we can implement this algorithm with only one scoped flood per node, we first explain how the algorithm works conceptually. Given the random permutation π on the nodes, we build the routing structure level by level, top-down. Hence, we start at level $i = \log \Delta$, with node $\pi(1)$ (*i.e.,* the first node in the random permutation). No message has been sent so far, and consequently this node is not covered on this level i (actually, on no level at all). Thus, node $\pi(1)$ start a flood with a hop count set to f_i. All nodes that receive this packet and are within r_i hops from node $\pi(1)$ will not flood on this level i anymore, as they are covered and become members of $\pi(1)$'s cluster. However, all nodes that receive the packet will remember the fact that node $\pi(1)$ is a beacon at level i. Then, we go through the nodes in the order given by the random permutation, and let the nodes flood f_i hops which were not covered before their turn comes. Once we are done with level i, we repeat the same process at level $i - 1$. That is, node $\pi(1)$ floods the network with hop count f_{i-1}, and nodes within r_{i-1} hops

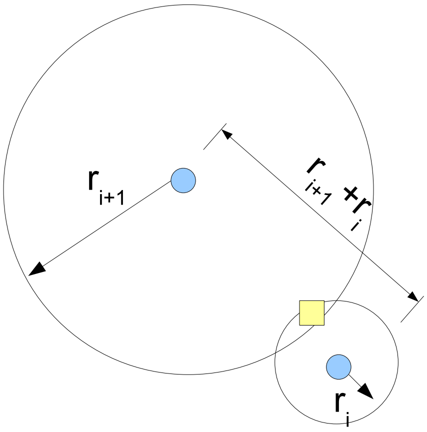


Figure 3.13: The flooding radius is chosen in such a way that beacons at level i hear the floods of beacons at level $i+1$. In static networks, this is $r_i + r_{i+1}$.

will not be elected beacons (*i.e.,* will not flood) at this level. Then, again, we go through the random permutation and let nodes flood f_{i-1} hops which were not covered before their turn comes. We continue this process until we reach level 1. Note that many floods are redundant, as for instance node $\pi(1)$ always floods first when we start the process on a new level. We can implement this process in a much more efficient way by letting all nodes flood only once. We explain this process below

When we do a single pass through the random permutation, every node u is a beacon at a given level $h(u)$. The flooding radius, however, will depend on the highest level at which a node is not covered. Let us denote by $h(u)$ the highest level at which node u is not covered. When node u's turn to flood comes, it will determine the value of $h(u)$ and call $flood(f_{h(u)}, h(u))$. A node v which receives this flood will determine the lowest level at which it could be a member of u's cluster, say $l(v)$. That is, it will determine the lowest value j for $l(v)$ such that $d(u,v) \leq 2^j$. This distance $d(u,v)$ is known since v just received a flood packet from u. It will then become a member of u's cluster for all levels above $l(v)$ for which it has no membership yet and are below $h(u)$. If a node becomes a member of u's cluster, it sends a *membership packet* (see Table 3.5) back to u. In this way, u learns the identifier of all nodes in its cluster. Note that u also applies this procedure to itself, and consequently could be a beacon at level i but not at level $j < i$. To summarize:

Pkt. Type	*Node Id.*	*Beacon Id.*	*Level*
$O(1)$ bits	$O(\log n)$ bits	$O(\log n)$ bits	$O(\log \Delta)$ bits

Table 3.5: Membership Packet

- Nodes start a scoped flood of the network with flood packets in a random order. The radius of the flood of a node u is determined by the highest level $h(u)$ at which this node has not yet been "covered" (or equivalently is not member of a cluster). This information is included in the packet.

- A node v that receives such a flood packet becomes a member of the sender's (say node u) cluster at level i (this concerns potentially more than one level) if the following conditions are met:

    - If node v joins u's cluster on level i, it is within r_i hops of u.
    - If node v joins u's cluster on level i, it is not already member of another cluster at this level.
    - i is lower than or equal to $h(u)$.
    - If node v joins u's cluster on level i, a membership packet is sent back to node u to confirm the membership.

In practice, this optimized scheme is equivalent to electing node $\pi(1)$ as a beacon on all levels as soon as it floods. Then, node $\pi(2)$ is elected as a beacon on all levels on which it was not covered by node $\pi(1)$, and so on. The other nodes infer from the hop count whether they consider a node that floods as a beacon on a given level.

The control traffic will be dominated by the messages sent back by nodes to beacons when they become members of a cluster, so should be rare. In other words, the election of *new* beacons, and consequently the transmission of membership packets, should be rare. However, we do not want the distance between nodes and beacons to grow by more than a constant factor. Since we assume that the maximum speed of the node is constant, the higher the level of a beacon, the more time it will take for nodes to double their distance to this beacon. We want to *elect* new beacons and update memberships only for levels at which the distances from beacons to cluster members could have been multiplied by a constant factor. Recall that the network is κ-constrained (see Definition 3.6). Consequently, the distance $d^{(t)}(u,v)$ between two nodes u and v cannot change by a factor κ in less than νd time steps (see Corollary 3.2). In particular, if a node is at distance 2^i of a beacon at the time it becomes a member of its cluster, then we have $d^{t+\nu 2^i} \leq \kappa 2^i$. Hence, we can update the memberships at level i only every $\nu 2^i$ time steps[8] (see Figure 3.14), and still have guarantees on distances to beacons. Finally, note that when new beacons are elected at level i, nodes which were previously beacons at

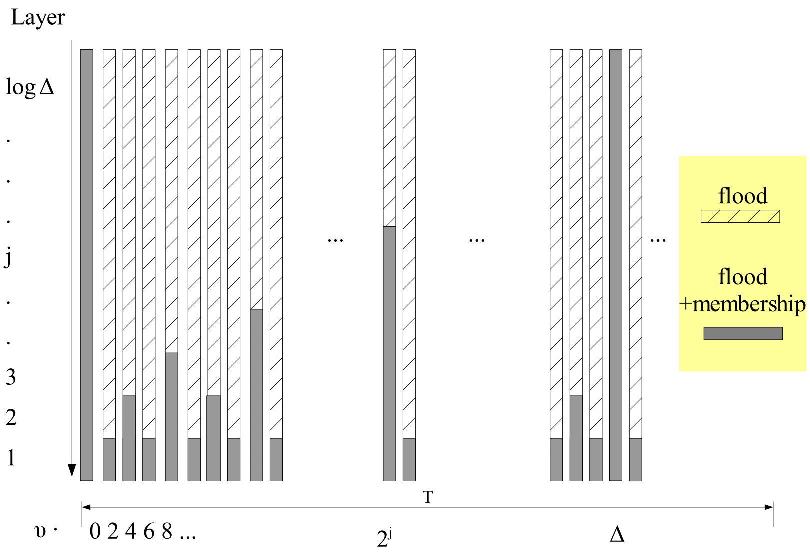


Figure 3.14: The memberships up to level i are updated every $\nu 2^i$ time steps. At the levels above, beacons elected at earlier time steps simply flood again.

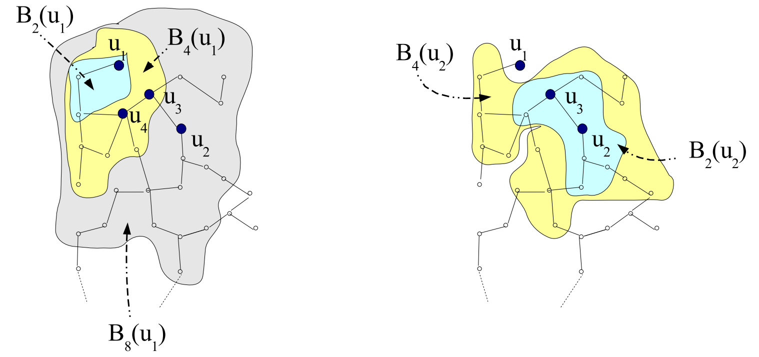
this level loose this status. This will lead to a routing scheme in which the distances can be distorted by at most a constant factor to be calculated in the sequel. Additionally, in a dynamic environment, routes can break. This is why we let the beacons (*i.e.,* the beacons elected in previous time steps) at all levels flood at every time step. Levels at which no membership updates take place simply use the floods of the beacons to update their routes toward theses beacons. This will ensure that a route always exists for all pair of nodes. We prove in Section 3.5 that our choice of flooding and cover radius guarantees that a route always exists for any pairs of nodes, and that the length of this route is within a constant factor of the shortest path. In Figure 3.15, we give a simple with three levels. The beaconing algorithm is presented in Algorithm 2. It is important to note that the routes are updated at every time step and consequently routing toward a beacon will always be successful. Further, when the membership at a given level i is updated, all the memberships at the levels $j < i$ will also be updated, and all memberships at these levels canceled.

3.4.2 Forwarding Algorithm

The forwarding algorithms works as follows: a source node u with a message for a target node v searches for v by first *probing* (see description of probing procedure at the beginning of the section) all the level 1 beacon it knows of. To

---

[8] Note that if a time step is a multiple of $\nu 2^i$, it is also a multiple of $\nu 2^j$, for $j < i$. Thus, we can again let every node up to level i flood exactly once at the appropriate level.



RT at u_2:

u_1	5	3	u_3

RT at u_3:

u_1	4	2	u_4
u_1	4	3	u_4

RT at u_3:

u_1	4	2	u_4
u_1	4	3	u_4
u_2	1	1	u_2

RT entry: (identifier) (distance) (level) (next hop)

Figure 3.15: The example start with empty routing tables. First, on the left, node u_1 floods at level 3. We focus on nodes u_2 and u_3. Node u_2 is within 8 hops from u_1 but further away than 4 hops. Consequently, it can only had an entry for node u_1 at level 3. At the same time, node u_3 can add an entry for node u_1 at the levels 2 and 3, since it is at distance 4 of u_1. Next, on the right, u_2's turn to flood comes (right after u_1's turn). This node is already covered at level 3. Consequently, it will flood at level 2. The node u_3 could potentially add an entry for this node at levels 1 and 2. However, it is already covered at level 2 and so adds only an entry for level 1. We do not show the entries beacons add for themselves.

```
input  : Routing Table at node $u$, Time t
output: -
Let $\Gamma = \max\{0 \leq j \leq \log \Delta | t \; mod(\nu 2^j) = 0\}$;
Clear routing table entries with $level \leq \Gamma$;
Let $h(u)$ be the level at which $u$ is a beacon;
if $\pi(\ell) = u$ then
    if $h(u) \leq \Gamma$ then
        Let $h(u)$ be the highest level at which $u$ is not covered;
        $h(u) = h(u)$;
    end
    $flood(f_{h(u)}, h(u))$;
end
```

Algorithm 2: Beaconing Algorithm at node u

do so, it looks at its routing table and selects all nodes it knows of at level 1. If all answers are negative, that is all the level 1 beacons within range reply with a success bit set to 0, node u probes all level 2 beacons it knows of. The procedure is repeated as long as all beacons answer negatively. A beacon at level i with an entry for the destination in its routing table does not answer directly to the source. Rather, this node will search downwards in the hierarchy by probing all the level $i-1$ beacons it knows of. We show in the next section that one of these beacons must have an entry for the destination. That beacon in turn probes all the beacons in knows of at level $i-2$. Meanwhile, the other beacons at level $i-1$ will answer negatively to the beacon at level i. The procedure is repeated recursively until the target itself is reached. The target will then answer to the source on the reverse path which will later be used for communication between the source and the destination. To summarize, the forwarding algorithm starts with an "upstream" phase during which the source node probes beacons level by level until a beacon is found which has the destination in its cluster. That beacon then starts a "downstream" phase, during which we go down in the hierarchy. We illustrate the forwarding procedure conceptually in Figure 3.16.

3.4.3 Load-balancing

This approach above guarantees a low network wide control traffic overhead. Over a long period of time all nodes will get approximately the same average overhead. Yet, beacons at the highest levels might get overloaded by the membership packets of the nodes in their cluster when a membership update takes place. Therefore, on rare occasions, these nodes will be hot spots in the network for a short period of time. To work around this problem, memberships can be distributed in the cluster instead of stored at the beacon itself. First, we now set the flooding radius to be $f'_i = \kappa(2r_{i+1} + r_i)$. Additionally, whenever a beacon floods at level i, it includes its membership at level $i+1$ in the packet. This information is stored by all nodes that receive this flood packet. This will guarantee that all nodes that are members of a cluster at level i, know how to reach all beacons at level $i-1$ inside that cluster. Now, assume that a node u becomes a member of the cluster of a beacon $b_i(u)$ at level i. Instead of sending the membership packet directly back to $b_i(u)$, it will now send its membership packet directly toward the beacon $\psi_{i-1}(u)$ at level $i-1$ inside this cluster, where $\psi_{i-1}(u)$ denotes the beacon inside $b_i(u)$'s cluster with the identifier closest to u's identifier. In turn, as soon as the packet reaches a node which is a member of $\psi_{i-1}(u)$'s cluster at level $i-1$, the membership packet is redirected toward the beacon $\psi_{i-2}(u)$ which is a member of $\psi_{i-1}(u)$'s cluster at level $i-1$ (*i.e,* the beacon inside $\psi_{i-1}(u)$'s cluster which has the identifier closest to node u's identifier). The process is repeated until we reach a single node, which will store u's identifier on behalf of $b_i(u)$. Note that the membership can only be registered at a single location in the cluster reachable through a unique sequence of clusters. This remains true even when nodes move. Indeed, the nodes in the cluster of $b_i(u)$ will only forward the

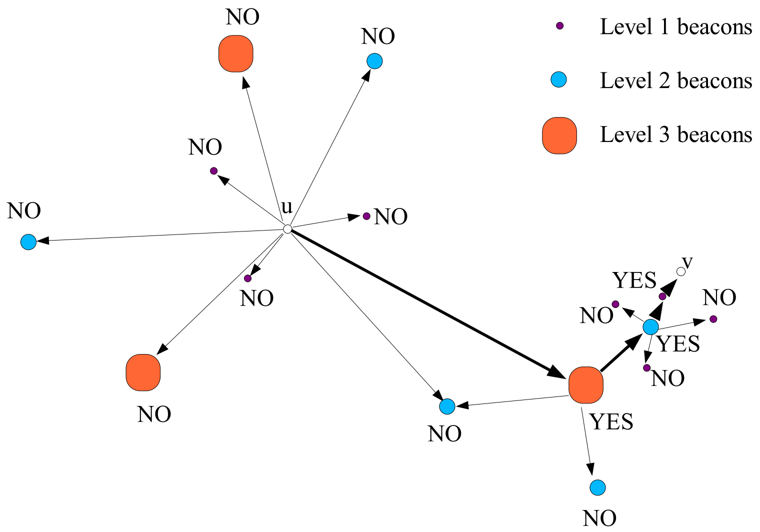


Figure 3.16: Node u has a packet for node v. It searches in its routing table for all beacons it knows of at level 1 and sends them a probe packet containing v's identifier. None of the beacons at level 1 has an entry for this node and consequently they all answer negatively to node u. Next, node u repeats the same procedure with all the beacons it knows of at level 2. Again, all beacons answer negatively. On the third level, now, a beacon has an entry for node v. This beacon will probe all the beacons it knows of at level 2, while the other beacons at level three will answer negatively to u. A beacon at level 2 must have an entry for v. This beacon again probes all the beacons it knows of at level 1 among which one must have an entry for v itself. Meanwhile, the other beacons reply negatively as they do not have any entry for v.

packet to beacons at level $i-1$ which were in the same cluster at the time the membership for this level got updated. Of course, whenever level $j < i$ is updated, we do now not only need to send u's identifier toward its new beacon at that level. Additionally, the node that holds u's identifier at level j might not be reachable anymore through a path of clusters with identifiers closest to u's. Consequently, this node will need to forward u's identifier toward the beacon at level j with the identifier closest to u's. Again the process will be repeated recursively until a single node is reached. As we will see, the cost of avoiding hot spots is a factor $\log n$ in the total control traffic. Finally and most importantly, with this procedure beacons no longer get overloaded. Rather, the traffic is be distributed in its cluster.

The data forwarding process remains the same except that the source node will not probe the beacon itself, but rather search for the node in the beacon's cluster that should hold the destination's identifier. If this node holds the identifier, it will then probe the beacons one level below in the same way. Recall that the nodes which potentially hold u's membership can be reached at any given instant in time through a unique sequence of clusters. The procedure is repeated until the destination is reached.

3.5 Performance Analysis

In this section, we analyze the performance of our algorithm analytically both in terms of control traffic and of route stretch. As in Section 3.4, we do this for a sequence of doubling and smooth connectivity graphs, and will use $\mathcal{G}^{(t)}(n, r_n)$ with USL mobility for illustration.

The bounds derived in this section hold *w.h.p.* when we are in a sequence of length n^ρ of α-doubling connectivity graphs. In the sequel, α, κ and ν are the constants derived in Section 3.3. Let us denote by $\Delta = O(\sqrt(\frac{n}{\log(n)}))$ the diameter of the network. To bound the control traffic necessary for beaconing, we will rely on the α-doubling property of the metric space to show that a node can only hear a constant number of beacons at every layer. We will first show that a ball of radius $2R$ around any node u can only contain at constant number of balls (clusters) of radius R, when we select the centers of the balls of radius R in an arbitrary order and ensure that two centers cannot be closer than R. We will later use this result to show that a node can hear at most a constant number of beacons at any given level.

Theorem 3.11 (Random Cover)**.** *Let $\mathcal{B}^X_{2R}(u)$ be a ball of radius $2R$ centered at u in a graph metric (X, d) with doubling constant α. Then, there exist at most $k \leq \alpha^2$ nodes v_i, $(i = 1, 2, .., k)$ such that $\mathcal{B}^X_{2R}(u) \subseteq \bigcup_i^k \mathcal{B}^X_R(v_i)$ and $min_{(i,j)} d(v_i, v_k) > R$.*

Proof. By definition of an α-doubling metric space, there must exist a cover of a ball of radius $2R$ consisting of at most α balls of radius R. Recursively, there must also exist an $\frac{R}{2}$-cover consisting of α^2 points. One can select at most one center v_i in each ball of radius $R/2$, as any other point inside this ball is within R of v_i. Hence, one can select at most α^2 such centers. ☐

Corollary 3.3. *Let B be a ball of radius $R > R'$ in an α-doubling metric space (X, d). Then, one can select at most $k \leq (\frac{R}{R'})^{2log(\alpha)}$ nodes v_i, $(i = 1, 2, .., k)$ such that $\mathcal{B}^X_R(u) \subseteq \bigcup_i^k \mathcal{B}^X_{R'}(v_i)$ and $min_{(i,j)} d(v_i, v_j) > R'$. In particular, if $R = \eta R'$ for some constant R, then k is at most a constant $(\eta)^{2log(\alpha)}$ independent of n.*

Proof. Let $R = 2^i R'$. Hence, R' is doubled $log\frac{R}{R'}$ times to obtain R. By Theorem 3.11, B can be covered by $\alpha^{2log\frac{R}{R'}} = (\frac{R}{R'})^{2log(\alpha)}$ balls of radius R'. ☐

Here, one can think of the radius R of the large balls as the flooding radius, and of the radius R' of the small balls as the cover radius. Indeed, we use this result to show that a node u can hear the floods of all beacons within a given radius R. Moreover, this ball of radius R can contain at most $(\frac{R}{R'})^{2log(\alpha)}$ beacons, since beacons must be at least R' apart.

3.5.1 Control Traffic

Theorem 3.12. *The average control traffic overhead per time step for beaconing is at most $O(n \log^2 n)$ bits.*

Proof. We will analyze the control traffic at level i. Recall that a beacon at level i floods a distance $f_i = \kappa(2^{i+1} + 2^i)$ at every time step. Further, at the time the memberships are updated at level i, a beacon node at this level *cannot* be within $r_i = 2^i$ of another beacon at that level. If it were the case, this node would not elect itself as a beacon at this level. Level i is updated every $\nu 2^i$ time steps. Consider a node u. By Corollary 3.10, no nodes that are further away than κf_i hops at the time the memberships are updated at level i could move within f_i of u in less than $\nu 2^i$ time steps. However, that is before this level is updated again. Consequently, the number of beacons whose flood can reach u at any given time step is at most the number of level i beacons in a ball of radius κf_i at the time the membership is updated. In turn, node u will broadcast[9] the flood packets of at most that many beacons for this level i. By Corollary 3.3, this number is a constant[10] given by $(\frac{\kappa f_i}{2^i})^{2log(\alpha)} = (3\kappa^2)^{2\log\alpha}$. Given that there are $O(\log n)$ levels, that there are n nodes and that a flood packet has size $O(\log n)$ bits, the average control traffic overhead per time step for beaconing is at most $O(n \log^2 n)$ bits. $\square$

We now compute the control traffic overhead necessary for nodes to update their memberships with beacons. Recall that level i and all levels below are updated every $\nu 2^i$ times steps and that a node can only be a member of one cluster at every level. Furthermore, a node only becomes a member of a cluster if it is within 2^i of the corresponding beacon.

Theorem 3.13 (Membership Update Overhead)**.** *The average control traffic overhead per time step to update memberships without load-balancing is at most*

$$\frac{n \log \Delta \log n}{\nu} = O(n \log^2 n)$$

bits.

Proof. Consider a sequence of T time steps. The memberships will be updated up to level i every $\nu 2^i$ time steps, so $\frac{T}{\nu 2^i}$ times in a sequence of length T. At the time of the update, a node can be at distance at most 2^i from a beacon at level i. Consequently, the overhead in bits generated by a node in a sequence of T time steps is upper bounded by $\sum_{i=1}^{\log\Delta} \frac{T}{\nu 2^i} 2^i \log n = \frac{\log\Delta}{\nu} \log n$. $\square$

---

[9] recall that when a node broadcasts a packet it is received by all direct neighbors in the connectivity graph. Consequently, there is one packet transmission per beacon of which a flood packet is received.

[10] In the load-balanced scheme, this constant is $(5\kappa^2)^{2\log\alpha}$.

Finally, we will show that the average control traffic overhead when load-balancing is used is increased by at most a factor $\log n$.

Theorem 3.14 (Membership Update Overhead)**.** *The average control traffic overhead per time step to update memberships with load-balancing is at most*

$$\frac{n \log^2 \Delta \log n}{\nu} = O(n \log^3 n)$$

bits.

Proof. Consider a sequence of T time steps. The memberships will be updated up to level i every $\nu 2^i$ time steps, so $\frac{T}{\nu 2^i}$ times in a sequence of length T. At the time of the update, a node can be at distance at most 2^{i+1} from a beacon at level $i-1$ inside its cluster at level i. Similarly, a node can be at distance at most 2^i from a beacon at level $i-2$ inside its cluster at level $i-1$. In the load balanced scheme, we have to count the overhead to go down the hierarchy of beacons. For a beacon at level i, this is at most $2^i \times 2$. Consequently, the overhead in bits generated by a node in a sequence of T time steps is upper bounded by $4 \sum_{i=1}^{\log \Delta} \frac{T}{\nu 2^i} 2^i \log n = 4 \frac{\log \Delta}{\nu} \log n$. However, node u is a member of a cluster at all $\log \Delta$ levels. Recall that the node that holds u's identifier must always be reachable through a path by choosing the beacon (cluster) with the identifier closest to u's. Hence, whenever level i gets updated, all $\log \Delta$ nodes that hold u's identity must follow the same procedure as u itself. We conclude that the overhead is upper bounded by $\log \Delta 4 \frac{\log \Delta}{\nu} \log n$ bits. ☐

3.5.2 Route Stretch

In this section we will show that the route found with the forwarding algorithm is only a constant factor longer than the shortest path. Additionally we show that the destination location discovery takes a negligible fraction of a flow throughput.

Theorem 3.15 (Routing Stretch)**.** *The worst case multiplicative routing stretch is $O(1)$.*

Proof. We first analyze the stretch without load balancing. Consider that we want to route from a node u to a node v, and that we had $2^k \leq d(u,v) \leq 2^{k+1}$, the last time level k was updated before the route search takes place. Let us denote by $b_i(v)$ the beacon to which node v had registered the last time level $i \leq k$ was updated before the route search takes place. Clearly, we have $d(u, b_v(k)) \leq \kappa(2^{k+1} + 2^k)$, and $d(b_i(v), b_{i-1}(v)) \leq \kappa(2^i + 2^{i-1})$. This is true since the membership of node v at level i must have been updated at most $\nu 2^i$ time steps before the routing takes place, and that at the time the time level i gets updated, we have $d(b_i(v), b_{i-1}(v))) \leq d(b_i(v), v) + d(v, b_{i-1}(v))$ by triangle inequality. Note that $d(b_i(v), b_{i-1}(v)) \leq f_{i-1}$ and that $d(u, b_v(k)) \leq f_k$. Hence, a route must exist between u and v and the length $r^{(t+\tau)}(u,v)$ of the route at time t is at most:

$$\begin{aligned} r^{(t)}(u,v) &\leq \sum_{i=1}^{k} f_k = \kappa \sum_{i=1}^{k} (2^{i+1} + 2^i) \\ &= 3\kappa \sum_{i=1}^{k} 2^i = 3\kappa \frac{2^{k+1}-1}{2-1} \leq 6\kappa d^{(t)}(u,v) \end{aligned}$$

In the worst cast, nodes u and v have moved closer together (by a factor κ) while the beacons have moved further apart. Indeed, we have $d^{(t+\tau)}(u,v) \leq \kappa d^{(t)}(u,v)$ for $\tau \leq \nu 2^k$ as our network is κ-constrained. Note that if we waited longer that $\nu 2^k$, memberships would be updated again at level k and we could find another beacon at distance 2^k at most from v at level k. Hence, the worst case stretch is $\frac{r^{(t+\nu 2^k)}(u,v)}{d^{(t+\nu 2^k)}} \leq 6\kappa^2 = O(1)$. □

Every node can only hear floods from a constant number ($\mu = (3\kappa^2)^{2\log\alpha}$, see Theorem 3.12) of beacons at every level. Recall that the source will first probe all beacons at level 1, then all beacons at level 2 and so on. The procedure is repeated up to level k at which the source u will send a packet to $b_k(v)$. Note that the distance from u to this beacon can be at most $\kappa 2^{k+1} + 2^k = f_k$ and so it must hear its floods. In turn, when routing down the hierarchy, beacon $b_j(v)$ will probe at most a constant number $((3\kappa^2)^{2\log\alpha}$ of beacons at level $j-1$. Finally, the distance between a node u and a beacon at level i can be at most f_i and a probe packet will traverse at most $2f_i$ packets when a beacon at level i is probed (back and forth). This means that for discovery of the location of the destination, we need a probe overhead of at most $\mu 6\kappa d(u,v)$ packet transmissions. Therefore, this is a negligible part of the throughput of a flow since it consumes roughly the equivalent of a few packet headers of a flow from source to destination. A similar statement can be made for the load-balanced case.

3.6 Implementation Issues

In Section 3.5, we have computed worst case bounds which may be conservative in terms of constants. In this section, we explore this aspect by looking at simulation results for the control traffic and for the stretch. Recall that we had computed that for each of the $O(\log n)$ levels, a node has to retransmit a packet of at most $(3\kappa^2)^{2\log\alpha}$ beacons. Even if we set the maximum speed as well as the parameter ν to 1, this is still $\sqrt{10}+20$ and consequently the constant in the bound on the overhead at least as high as $(3(\sqrt{10}+20)^2)^2 \approx 2.5 \cdot 10^6$! In Figure 3.17, however, we show that in practice this constant is approximately 30. This simulation was run with 50 up to 10000 nodes moving at a maximum speed of 1. One can observe that the experimental scaling behavior corresponds extremely well to the theoretical behavior. To stress this fact, we also plot $100 \log n$ as a benchmark. Note that the overhead is expressed in number of packets rather than bits (a packet being of size $O(\log n)$).

Similarly, in Figure 3.18 we show that for a network of 1000 nodes, the stretch is at most 1.5 for all node pairs. If we compute the maximum theoretical

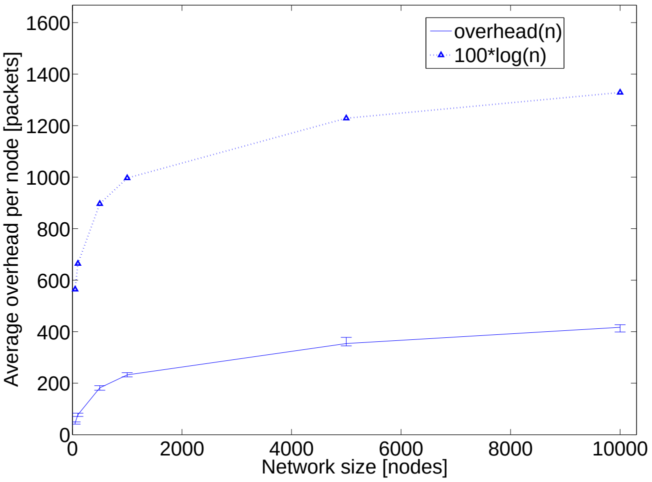


Figure 3.17: Average control traffic overhead per node in packets as a function of the network size. Nodes move at a speed of maximum speed of 1. The confidence interval is given by the 95% and 5% percentiles. The size of a packet is $O(\log n)$ bits. We also plot $100 \log n$ to show that our analytical predictions match the simulation results.

stretch, we can show that it is again considerably larger and hence a pessimistic bound. These small constants could make a practical implementation realistic.

We have made a certain number of assumptions in our models, which we now clarify. In practice, the random permutations on the nodes, which determines the order in which the flooding occurs, could be implemented by using random timers; more precisely, by letting all nodes draw a random delay independently of each other every ΔT seconds. Obviously, the interval from which nodes draw this delay should be made sufficiently large so that we can avoid collisions. However, a level in the hierarchy will be rapidly covered, and in a practical implementation the covers at different levels could be built in parallel. Further, different parts of the network are independent except at the highest level, and we could exploit this spatial diversity to parallelize the beaconing process. Hence, we speculate that it is possible to reduce the length of the beaconing phase to a small constant times the maximum round-trip time. Note that one could apply the algorithms to underlying networks that are not doubling. In this case, we would not be able to give provable bounds on the control overhead and the stretch as we did for doubling networks.

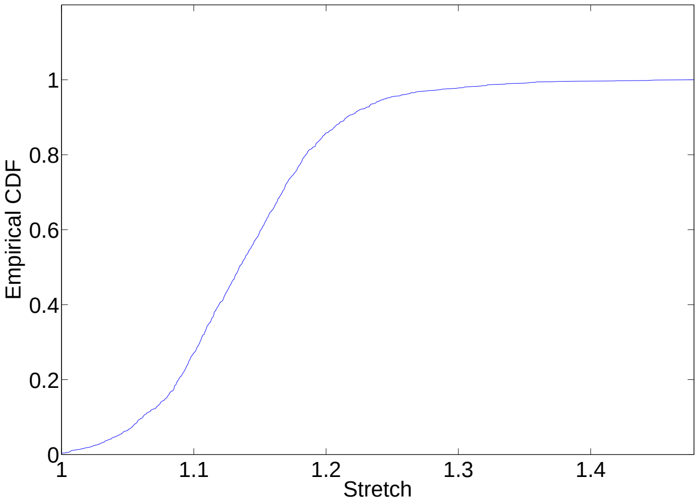


Figure 3.18: Empirical cumulative distribution of stretch (route length/shortest path) for a network with 1000 nodes moving at a maximum speed of 1.

3.7 Concluding Remarks

In this chapter, we showed that a large class of wireless network models belong to a larger class of networks, the *doubling* networks, in which efficient routing can be achieved. Hence, we have formalized the intuition developed in Chapter 2, that the structure of wireless networks was intrinsically low-dimensional. To design an efficient routing scheme, one can hierarchically decompose the network by relying on the doubling property to prove that the control traffic overhead and the stretch will remain low, even for dynamic doubling networks. This holds for a fairly broad class of uniform speed-limited (USL) mobility models. One advantage of the proposed routing algorithm is that it is robust, in that it works well in certain situations in which other existing algorithms cannot work well. This was illustrated in Section 3.2.1 for an example network with obstacles. We believe that many more such examples can be created where the use of the doubling rather than geographic properties would be crucial. To the best of our knowledge, our results are the first provable bounds for routing quality and costs for dynamic wireless networks. These techniques might give us insight into algorithm design for more sophisticated wireless network models.

Comparison-based Nearest Neighbor Search

4

We now move on to the second main topic of this thesis, namely comparison-based search in databases. In this chapter, we address this search problem from a formal theoretical point of view. In particular, we investigate methods to search a database of objects that do not necessarily live in a metric space. Recall that we consider the situation where we want to search and navigate a database, but we do not know the underlying relationships between the objects. This implies, in particular, that distances may be difficult to discern, or may not be well-defined. As mentioned in Chapter 1, such situations are common with objects where human perception may be involved. A collection of pictures of faces, taken from different angles and distances is an illustration of such a dataset. Indeed, the distances between feature vectors might be far from the similarity perceived by humans. Notwithstanding, either with human-assistance or approximate classification, we may be able to determine the relative proximity of an object with respect to a small number of other objects[1]. More precisely, Humans have the ability to compare objects and make statements about which are the most similar ones, though they can probably not assign a meaningful numerical value to similarity. This led to the question of how to design search algorithms based on binary similarity decisions of the type “A looks more like B than C”.

More formally, we aim to design an algorithm that given a query object (*e.g.,* the face of a person we are looking for), efficiently returns an object that is similar to that object among the objects in a database. To do so, we have access to a similarity oracle which, given two reference objects and a query object, can tell which of the two reference objects is most similar to the query object. We measure the performance of all our algorithms in terms of

---

[1] We provide the architecture for a practical implementation of such a system in Chapter 5.

the number of questions that we need to ask the oracle. This is motivated by the fact that we consider that it is very costly to ask human users to answer questions. Hence, we aim at minimizing the number of such questions. We can pre-process the database during a learning phase, and use the resulting answers to facilitate the search process.

One way to understand this setup is to consider that the objects live in a hidden space, which can only be accessed through the aforementioned oracle. We do *not* make the assumption that the "hidden" space in which the database objects live needs to be a metric space. Using the aforementioned oracle, one can retrieve for every object u in the database a sorted list of the other objects according to their distance to u. We call the position of object v in this list the *rank* of v with respect to u, and denote it by $r_u(v)$. Clearly, this relationship can be asymmetric *i.e.,* $r_u(v) \neq r_v(u)$ in general. This setup raises several new questions and issues, as any space can be described by its ranks relationships. How much does the fact that the rank of some object v w.r.t. some other object u is k, and the rank of w w.r.t. u is k' tell us about the rank of w w.r.t. v? In this chapter, we introduce the notion of *rank distortion* (see Section 4.2 for a rigorous definition). The rank distortion captures how closely $r_v(w)$ is related to the average $\frac{1}{n}\sum_u |r_u(v) - r_u(w)|$. The framework introduced in [GLS08], defines approximate triangle inequalities on the ranks, another way to capture these relationships. Those inequalities roughly tell us how "transitive" the similarity relationship is and give us a notion of *combinatorial disorder.* If we have this information, we can use partial rank information to estimate, or infer the other ranks. In this chapter, we will first investigate the case where we can use such a characterization of the hidden space as an input to our algorithms. We develop a randomized hierarchical scheme that improves the existing bounds for nearest neighbor search based on a similarity oracle (see Section 4.1). We also prove, as far as we know, the first lower bound on the average number of questions to be asked for randomized nearest-neighbor search in this setup (see Section 4.4). Then, in Section 4.5, we ask what can be done if no characterization of the hidden space is known and therefore cannot be used as an input to the algorithms. In that case, we cannot estimate, or limit, ranks anymore if we have partial rank information. Nevertheless, we develop algorithms that can decompose the space such that dissimilar objects are likely to get separated, and similar objects have the tendency to stay together. This generalizes the notion of randomized k-d-trees (see [DF08]) to our setup. Building on this intuition, we introduce the notion of *rank-sensitive hashing* (RSH) in Section 4.5.3. Similarly to locality-sensitive hashing, we can retrieve one of the R nearest neighbors of a query point very efficiently. The hash function itself does not require any characterization of the subjacent space as an input. However, the smallest value of R we can choose depends on the rank distortion. In general, both the criteria (combinatorial disorder and rank distortion) we use to characterize the hidden space seem to capture how "homogeneous" that space is. It appears that the less homogeneous it is, the more difficult it becomes to search. In particular, if the rank relationship is very asymmetric, and some objects are far from every other object, the information contained about those objects in the ranks matrix is very sparse and hard to capture. We apply this idea of RSH to NN search, but we believe that this might be useful in other scenarios as well. We investigate the implications of RSH for objects randomly placed in $\Re^d$.

4.1 Relationship to Published Works

The nearest neighbor (NN) problem, and many variations thereof, have been extensively studied in the literature (see for instance [Cla06] and [Ind04] for surveys). In particular, very efficient algorithms have been developed for specific classes of metric spaces, such as metric spaces with a low intrinsic dimension or a bounded growth factor. In [KL04], the authors introduce ϵ-nets, a very simple data structure for nearest neighbor search (and many other applications). The complexity of those nets depends on the doubling dimension (see definition 3.2, in Section 3.2) of the underlying space. In [KR02], the authors present a random sampling algorithm to produce a data structure for search in growth restricted metrics. The restricted growth guarantees that a random sample will have some nice properties. In particular, by randomly selecting a small number of representatives at different scales for every object in a learning phase, one can zoom in on the nearest neighbor of a query point during the search phase. On the other hand, search when the underlying space is not necessarily a metric space appears to have very little prior work. In some sense, it is a generalization of the above problem, as any dataset can be represented by its rank relationships.

The problem of searching with a similarity oracle was first studied in [GLS08][2], where a random walk algorithm is presented. The main limitation of this algorithm is the fact that all rank relationships need to be known in advance, which amounts to asking the oracle $O(n^2 \log n)$ questions, in a database of n objects. The authors of [LZ09] and [GLS08] work with a combinatorial framework for nearest neighbor search, which defines approximates inequalities for ranks analogous to the triangle inequality for distances. Their bounds depend crucially on the combinatorial disorder, represented by the *disorder constant* D of the database (a notion to be defined more formally in Section 4.2, which captures to what extent the triangle inequality on ranks can be violated). In [LZ09], a data structure similar in spirit to ϵ-nets of [KL04] is introduced. It is shown that a learning phase with complexity $O(D^7 n \log^2 n)$ questions and a space complexity of $O(D^5 n + Dn \log n)$ allows to retrieve the nearest neighbor in $O(D^4 \log n)$ questions, in a database of n objects. The learning phase builds a hierarchical structure based on coverings of exponentially decreasing radii[3]. We will show (see Section 4.3) that we can improve those bounds by a factor polynomial in D, if we are willing to accept a negligible (smaller than $\frac{1}{n}$) probability of failure. Our algorithm is based on random sampling, and hence can

---

[2] Our interest in this formulation arose from an applied viewpoint in the implementation of the facebrowser system (see Chapter 5).

[3] the radius of a ball is defined as the cardinality of that ball.

be seen as a form of metric skip list (as introduced in [KR02]), but applied to a combinatorial (non-metric) framework. However, the fact that we do not have access to distances forces us to use new techniques in order to minimize the number of questions we need to ask (or ranks we need to compute). In particular, we sample the database at different densities, and infer the ranks from the density of the sampling, which we believe is a new technique. We also need to relate samples to each other when building the data structure top down. We also present what we believe is the first lower bound for our problem of searching through comparisons.

A natural question to ask is whether one can develop data structures for NN when a characterization of the underlying space is unknown. This has been addressed in the case when the underlying metric space has low "intrinsic" dimension and one has access to metric distances in [KL04, DF08]. In [DF08], it is shown that one can build a binary tree decomposition of a dataset of points in $\Re^d$, such that the diameter of the sets in the tree is reduced by a constant after a number of level that only depends on the intrinsic dimension of the data, and not d. The term intrinsic dimension either refers to the doubling dimension (also referred to as Assouad dimension) or the local covariance dimension[4]. Therefore, one can similarly ask such a natural question in our framework where we do not have access to metric distances (or they do not exist). We develop a binary tree (hierarchical) decomposition, when the characteristics of the underlying space (disorder constant) is unknown. This extends the result of [DF08] to our framework, where we only have access to the underlying space through comparisons.

The approximate nearest neighbor problem consists in finding an element that is at distance at most $(1+\epsilon)d_{min}$ from the query point q, where $d_{min} = \min_i d(i,q)$. In [IM98], Indyk and Motwani present two algorithms for this problem. In particular, *locality sensitive hashing*, through which they obtain an algorithm with polynomial learning and query time polynomial in d and $\log n$. For binary vectors, it is remarkable that the performance of the algorithm does not depend on the dimension. A survey of results for LSH can be found in [AI08]. In [Pan06], Panigrahy shows that instead of using a large number of hash tables as it is the case in the approach above, only a few can be used. These are then hashed to several randomly chosen objects in the neighborhood of the query point, and it is shown shows that at least one of them will fall into the same bucket as the nearest neighbor. The authors of [MNP06] prove a lower bound on the parameter $\rho = \frac{\log 1/p}{\log p/P}$ for (r, cr, p, P)-locality sensitive hashing schemes. We present a new hashing scheme that is *rank-sensitive* (RSH). How efficient the scheme is depends on another property of the hidden space, its *rank-distortion*. The rank-distortion need not be an input to the algorithm, however, the performance will depend on it. We give sufficient conditions for RSH to work and demonstrate its application to NN search. We also evaluate its performance for randomly placed points in $\Re^d$ and show that its performance

---

[4] Set $S \subset \Re^D$ has local covariance dimension (d, ϵ, r) if its restriction to any ball of radius r has covariance matrix whose largest d eigenvalues satisfy $\sigma_1^2 + ... + \sigma_d^2 \geq (1-\epsilon)\sum_{i=1}^{D} \sigma_i^2$.

improves with d.

To the best of our knowledge, the notion of rank-sensitive hashing and approximate (and randomized) nearest neighbor search using similarity oracle is studied for the first time in this thesis. Moreover, the hierarchical search scheme proposed is more efficient than earlier schemes. The lower bound presented appears to be new and demonstrates that our schemes are (almost) efficient.

4.2 Definitions and Problem Statement

In this section, we define formally the notions that we use in the rest of the chapter. We consider a hidden space $\mathcal{K}$ with distance function $d(.,.)$, and a database of objects $\mathcal{T} \subset \mathcal{K}$, with $|\mathcal{T}| = n$. We do not have access to the distances between the objects in $\mathcal{K}$ directly. We can only access this space through a *similarity oracle* which for any point $q \in \mathcal{K}$, and objects $u, v \in \mathcal{T}$ returns:

$$\mathcal{O}(q, u, v) = \begin{cases} u & \text{if } d(u,q) \leq d(v,q) \\ v & \text{else} \end{cases} \tag{4.1}$$

For the sake of simplicity, we consider that all distances in $\mathcal{K}$ are different. Note that the objects do not need to be in an underlying metric space for this similarity oracle. We now define the notion of *rank*.

Definition 4.1. *The rank of u in a set $\mathcal{S}$ with respect to v, $r_v(u, \mathcal{S})$ is equal to c, if u is the c^{th} nearest object to v in $\mathcal{S}$.*

To simplify the notation, we only indicate the set if it is unclear from the context *i.e.,* we write $r_v(u)$ instead of $r_v(u, \mathcal{S})$ unless there is an ambiguity. Note that rank need not be a symmetric relationship between objects *i.e.,* $r_u(v) \neq r_v(u)$ in general. Further, note that we can rank m objects w.r.t. an object o by asking the oracle $O(m \log m)$ questions. To do so, create the ranking w.r.t. o by adding one object at a time. Observe that in order to add the $(i+1)^{th}$ object to the list, we need to ask $\log(i)$ questions. More precisely, we need to ask whether the $(i+1)^{th}$ object is closer to o than the object currently at position $i/2$. Then, we can recurse on the set new set of objects (*e.g.*, if the object to insert is closer than the $i/2^{th}$ object, select the $i/4^{th}$ object as the new "pivot"). Summing over i, the total number of questions to be asked to sort m objects is $O(m \log(m))$.

Our first characterization of the space of objects is through a form of approximate triangle inequalities first introduced in [LZ09] and [GLS08][5]. Instead of defining a relationship between distances, these triangle inequalities define a relationship between ranks. These relationships depend on a property of the space called the *disorder constant* D. In [LZ09] and [GLS08], four such inequalities are defined, all implying the others with $D' = D^2$.

---

[5] We have another characterization called rank distortion in Definition 4.3.

Definition 4.2. *The rank disorder of a set of objects S is the smallest D such that $\forall x, y, z \in S$, we have the following approximate triangle inequalities:*

1. $r_x(y, S) \leq D(r_z(x, S) + r_z(y, S))$
2. $r_x(y, S) \leq D(r_x(z, S) + r_y(z, S))$
3. $r_x(y, S) \leq D(r_x(z, S) + r_z(y, S))$
4. $r_x(y, S) \leq D(r_z(x, S) + r_y(z, S))$

In particular, $r_x(x, S) = 0$ and $r_x(y, S) \leq D r_y(x, S)$.

A rank-ball around some point x is defined as $\beta_x(r) = \{i \in S | r_x(i) \leq r\}$. Recall that a "distance" ball is defined as $\mathcal{B}_u(r) = \{i \in S | d(u, i) \leq r\}$. Hence, if $r_x(v) = r$ and $o \in \beta_x(r)$, then $o \in \mathcal{B}_x(d(x, v))$.

We further define the rank matrix $\mathcal{R}$ where $r_{ij} = r_i(j)$, and the matrix $\mathcal{W} = \mathcal{R} + \mathcal{R}'$ (note that the matrix $\mathcal{W}$ is symmetric). For a subset $S \in \mathcal{K}$, we define its diameter $\Delta_S = \max_{i,j \in S} w_{ij}$. Let ρ_i denote the i^{th} column of $\mathcal{R}$ *i.e.*, we associate to every object $o \in \mathcal{T}$ a vector $\rho_o \in \{0, ..., n-1\}^n$, such that the j^{th} coordinate of o is given by $r_j(o)$.

We now define the *rank-distortion* of a set S as follows:

Definition 4.3. *We say of a set of objects S that its rank distortion function is $f : \mathbb{N}_+ \rightarrow \Re_+$, if f is monotonically increasing and if there exists $\gamma > 0$ (the* rank-distortion*) such that $\forall u, v \in S$:*

$$f(r_u(v)) \leq ||\rho_v - \rho_u||_1 \leq \gamma f(r_u(v))$$

Lemma 4.1. *If the function f is linear* i.e., *$f = c r_u(v)$, then the four approximate triangle inequalities are implied with $D \leq \gamma$.*

For example, for the first inequality, we have $r_x(y, \mathcal{K}) \leq ||\rho_x - \rho_y||_1 / c \leq (||\rho_x - \rho_z||_1 + ||\rho_z - \rho_y||_1)/c \leq \gamma(r_z(x, \mathcal{K}) + r_z(y, \mathcal{K}))$. The proof for the other inequalities is similar.

We can define the nearest neighbor problem as follows:

Definition 4.4 (R-nearest neighbor problem)**.** *Given a set of objects $\mathcal{T}$ and a query point q, return one of the R objects in $\mathcal{T}$ closest to q. In particular, if $R = 1$, return the closest object to q in $\mathcal{T}$.*

We say that a hashing scheme is (r, R, p, P)-sensitive if

Definition 4.5. *We call a hashing scheme h, "(r, R, p, P)-rank-sensitive" if $\forall q \in \mathcal{K}, u \in \mathcal{T}$,*

$$\mathcal{P}\left[h(q) = h(u) | r_q(u, \mathcal{T}) < r\right] > p \text{ and } \mathcal{P}\left[h(q) = h(u) | r_q(u, \mathcal{T}) > R\right] < P$$

Note that we should have $P < p$.

Finally, we say that a result holds **with high probability (w.h.p.)** if it hold with probability higher than $1 - \frac{1}{n}$.

4.3 Contributions

One of the difficulty of searching a hidden space arises from the fact that we cannot know how transitive the rank relationship is *i.e.,* we cannot know whether the fact that A is similar to B, and B is similar to C implies that A is similar to C. This is problematic in the sense that even if the oracle tells us that A is closer to our query point than B, it does not necessarily imply that points close to A are better candidates than points close to B. In metric spaces, such a guarantee is provided by the triangle inequality. A way to characterize the hidden space is to limit the extent to which the triangle inequality on ranks can be violated. The combinatorial framework, introduced in [LZ09, GLS08], (see definition of approximate triangle inequalities) does exactly that. In this chapter, we improve on their results in two ways. We provide more efficient algorithms using randomization and also provide a new lower bound for such randomized algorithms. More precisely, we show that if we only require success with high probability for nearest neighbor search, we can exploit the fact that a sample of randomly chosen points will have nice properties. In particular, it will be very likely that every object in the database will have an object sampled that is similar to itself. By sampling more and more densely at every level of a hierarchy, we will ultimately sample all objects. The key observation is that in order to find the sample closest to a particular object, we will only need to look at objects for which the closest sample at the level above in the hierarchy was also close to that object. We introduce a conceptually simple randomized hierarchical scheme that allows us to reduce the learning compared to the existing algorithm (see [LZ09, GLS08]) by a factor D^4, memory consumption by a factor $D^5/\log^2 n$, and a factor $D/\log n \log\log n^{D^3}$ for search (see Section 4.1). This algorithm's performance is best when the disorder constant is small.

Theorem 4.1. *There exists a data structure, which for a given query point q, can retrieve its nearest neighbor with high probability in* $O(D^3 \log^2 n \log\log n^{D^3})$ *questions. The learning requires asking* $O(nD^3 \log^2 n \log\log n^{D^3})$ *questions in total. We need to store* $O(n \log^2 n/\log(2D))$ *bits in total.*

We then prove a lower bound on the average search time to retrieve the nearest neighbor of a query point for randomized algorithms. Our result confirms the intuition we have developed so far. Indeed, the higher the disorder constant, the more difficult it becomes to search. One way to interpret this result is that the higher the disorder constant D, the less information the answer to a question to the Oracle provides us.

Theorem 4.2. *There exists a space, a configuration of a database of n objects in that space and a distribution over placements of the query point q such that no randomized search algorithm, even if* $O(n^3)$ *questions can be asked in the learning phase, can find q's nearest neighbor in the database for sure (with a probability of error of 0) by asking less than an expected* $\Omega(D\log(\frac{n}{D^2}) + D^2)$ *questions.*

Consequently, our schemes are asymptotically (for n) within a factor $\tilde{O}(D)$ of the optimal scheme (*i.e.,* within $Dpolylog(n)$ questions of the optimal search algorithm). The proofs of those two theorems are provided in Section 4.4.

Clearly, one of the limitations of the schemes above is that we need to know the disorder constant. It might be possible to estimate the value of the disorder constant based on a sample of objects in the database. Limitations of this approach are the fact that we might considerably degrade the performance of the algorithms if the estimator is inaccurate, and that we might run into trouble if the query point does not come from the same distribution as the database points $\mathcal{T}$. We therefore extend, in Theorem 4.6, the idea of k-d-trees to our setup. We provide an algorithm to build a binary tree that adapts to the disorder of the hidden space (see [DF08] for an analogous result for $\Re^d$). In Section 4.5.3, we present a new *rank-sensitive* hash function with many potential applications. The idea of rank-sensitive hashing is that by computing many times a hash function drawn at random, similar objects will be assigned the same hash value more frequently than dissimilar objects. The performance of the rank-sensitive hashing scheme depends on the rank-distortion of the hidden space. Instead of capturing how "transitive" the rank relationship is, the rank disorder captures how the rank $r_u(v)$ relates to the average rank *i.e.,* $\mathbb{E}\left[|r_j(v) - r_j(u)|\right]$. In other words, if we picked an object x at random, and sorted all other objects w.r.t. this object, how would $|r_x(v) - r_x(u)|$ relate to $r_u(v)$? If $r_u(v)$ can be approximated by a function f of $\mathbb{E}\left[|r_j(v) - r_j(u)|\right]$), then we can exploit this fact to separate points close to q and points far from q.

Theorem 4.3. *Given a set of objects S with rank-distortion function f, and rank distortion γ, there exists a function h which is $(r, (1+\epsilon)r, 1 - \frac{f(r)}{n^2}, 1 - \frac{f((1+\epsilon)r)}{n^2\gamma})$-rank-sensitive.*

A special case is when the function f is constant. Then, the behavior of the function is similar to the one observed with locality-sensitive hashing for binary vectors. One of the consequences is that we can retrieve one of the $R = (1+\epsilon)r$ nearest neighbors of a query point q in $n^{O(\frac{\gamma}{\epsilon})}$ questions. By using the output of the hash function in a different way, we can compute an overall ranking of the objects. We can then retrieve "popular" objects *i.e.,* those which are close to many other objects. This idea is discussed in Section 4.5.2.

4.4 Searching with Known Disorder Constant

In this section, we make the assumption that the disorder constant D, of $\mathcal{T} \cup \{q\}$ is known, and that we can consequently use it as an input to our algorithms. Knowing D is an advantage, as it allows one to rapidly exclude some candidate objects during the search phase. In other words, we can take advantage of the fact that if we found an object close to the query point q, then objects which are far from that object cannot be the nearest neighbor of q. We first present an algorithm for nearest-neighbor search. The algorithms builds a hierarchical decomposition of the test set $\mathcal{T}$. The construction succeeds with high probability *i.e.,* for a fixed query point q, the data structure is such that it will return q's nearest neighbor w.h.p. Then, we present a lower bound on the search complexity.

4.4.1 Hierarchical Data Structure For Nearest-Neighbor Search

The learning phase is described in Algorithm 3. The algorithm builds a hierarchical decomposition level by level, top-down. At each level, we sample objects from the database. The set of samples at level i is denoted by S_i, and we have $|S_i| = m_i = a(2D)^i \log n$, where a is a constant independent of n and D. At each level i, every object in $\mathcal{T}$ is put in the "bin" of the sample in S_i closest to it. To find this sample at level i, for every object o we rank the samples in S_i w.r.t. o (by using the oracle to make pairwise comparisons). However, we will show that given that we know D, we only need to rank those samples that fell in the bin of one of the at most $4aD \log n$ nearest samples to o at level $i-1$. This is a consequence of the fact that we carefully chose the density of the samples at each level. Further, the fact that we build the hierarchy top-down, allows us to use the answers to the questions asked at level i, to reduce the number of questions we need to ask at level $i+1$. This way, the number of questions per object does not increase as we go down in the hierarchy, even though the number of samples increases. The search process is described in Algorithm 4. The key idea is that the sample closest to the query point on the lowest level will be its nearest neighbor. Hence, by repeating the same process as for inserting objects in the database, we can retrieve the nearest neighbor w.h.p.

We will now show that Algorithm 3 succeeds with probability higher than $1 - \frac{1}{n}$ (w.h.p.) and that it requires asking less than $O(D^3 \log^2 n \log\log n^{D^3})$ questions w.h.p.

Theorem 4.4. *Algorithm 3 succeeds with probability higher than* $1-\frac{1}{n}$ *(w.h.p.) and it requires asking less than* $O(nD^3 \log^2 n \log\log n^{D^3})$ *questions w.h.p.*

We first prove two technical lemmas that we will need to prove Theorem 4.4.

Lemma 4.2. *If we throw* $m = ab \log n$ *balls into* b *bins, each chosen uniformly at random, then the first bin will contain at least one ball with probability* $1-\frac{1}{n^a}$

Proof. The probability that a bin contains no ball is

$$\begin{aligned} \mathcal{P}\left[\text{a bin contains no ball}\right] &= (1-\tfrac{1}{b})^{ab\log n} \\ &\leq e^{-a\log n} \\ &= \tfrac{1}{n^a} \end{aligned}$$

□

```
input  : A database with $n$ objects $z_1, ..., z_n$ and disorder $D$
output: For each object $u$, a vector $\phi_u$ of length $\log n/\log(2D)$. The
         list of all samples $\cup_i S_i$
for $i \leftarrow 1$ to $L = \frac{\log n}{\log 2D}$ do
    Let $S_i$ be a set of $a(2D)^i \log n$ objects chosen u.a.r. in the
    database $\mathcal{T}$;
    for $j \leftarrow 1$ to $n$ do
        if $i = 1$ then
            $c_j(1) \leftarrow S_1$
        else
            $c_j(i) \leftarrow$
            $\{v \in S_i |\text{rank of } \phi_v(i-1) \text{ in } c'_j(i-1) \text{ less than } 4aD\log n\}$;
            /* $c_j(i)$ is the set of samples in $S_i$, for which
               the closest sample in $S_{i-1}$ was one of the (at
               most) $4aD\log(n)$ closest sample to $z_j$ in $S_{i-1}$
               */
        end
        if $|c_j(i)| = 0$ then
            Report Failure
        else
            $c'_j(i) \leftarrow$ sort $c_j(i)$ according to $r_{z_j}(v, S_i)$, $\forall v \in c_j(i)$;
            $\phi_j(i) \leftarrow$ first object in $c'_j(i)$;
            /* $\phi_j(i)$ is the sample in $S_i$ closest to $z_j$      */
        end
    end
end
```

Algorithm 3: Learning Algorithm

```
input  : A database with $n$ objects and disorder $D$, the list of
         samples, the vectors $\phi$, a query point $q$
output: The nearest neighbor of $q$ in the database
$c'_q(1) = S_1$;
for $i \leftarrow 2$ to $L = \frac{\log n}{\log 2D}$ do
    $c_q(i) \leftarrow$
    $\{v \in S_i |\text{position of } \phi_v(i-1) \text{ in } c'_q(i-1) \text{ smaller than } 4aD\log n\}$;
    $c'_q(i) \leftarrow$ sort $c_q(i)$ according to $r_q(v, S_i)$, $\forall v \in c_q(i)$;
end
return first object in $c'_q(\frac{\log n}{\log 2D})$
```

Algorithm 4: Search Algorithm

Lemma 4.3. *We throw m balls into n bins, each chosen uniformly at random. We number the bins from 1 to n. Then, the probability that the bins 1 to $\frac{n}{c}$ contain more than $(1+\tau)m/c$ or less than $(1-\tau)m/c$ balls is at most $2e^{-\tau^2 m/3c}$.*

Proof. We throw the balls one after the other into the bins. Let $X_i = 1$ if the i^{th} ball falls in one of the $\frac{n}{c}$ first bins, and 0 else. Let $X = \sum_i X_i$. Clearly, we have $\mathbb{E}[X] = m/c$, as $\mathcal{P}[X_i = 1] = 1/c$ and all X_i's are independent. By the Chernoff Bound (see for instance [MU05], page 67), we have $\mathcal{P}[|X - \mathbb{E}[X]| > \tau m/c] < 2e^{-\tau^2 m/3c}$. □

We are now ready to prove Theorem 4.4.

Proof of Theorem 4.4. Let $m_i = a(2D)^i \log n$ denote the number of objects we sample at level i, and let S_i be the set of samples at level i *i.e.*, $|S_i| = m_i$. Here, a is an appropriately chosen constant, independent of D and n. Further, let $\lambda_i = \frac{n}{(2D)^{i-1}}$. We will first show the for every object $o \in \mathcal{T} \cup \{q\}$, where q is the query point, the following four properties of the data structure are true w.h.p.

1. $|S_i \cap \beta_o(\lambda_{i+1})| \geq 1$
2. $|S_i \cap \beta_o(\lambda_i)| \leq 4aD \log n$
3. $|S_{i+1} \cap \beta_o(\lambda_{i-1})| \leq 16aD^3 \log n$
4. $|S_i \cap \beta_o(4\lambda_i)| \geq 4aD \log n$
5. $|S_{i+1} \cap \beta_o(4\lambda_{i-1})| \leq 64aD^3 \log n$

Fix an object o and a level i. To visualize the proof, place all objects in the database on a line, such that the object u with rank $r_o(u, \mathcal{T}) = r$ is located at distance r from o (see figure 4.1). Property 1 tells us that at least one of the samples at level i will be such that its rank w.r.t. o is smaller than λ_{i+1} *i.e.*, $\exists s \in S_i$ s.t. $r_o(s) \leq \lambda_{i+1}$. Clearly, by Lemma 4.2, this is true with probability at least $1 - \frac{1}{n^a}$ (set $m = m_i$ and $b = (2D)^i = \frac{n}{\lambda_{i+1}}$ in the lemma). Property 2 tells us that not too many objects can have rank less than λ_i at level i w.r.t. o. Let $c = \frac{n}{\lambda_i} = (2D)^{i-1}$. Now, by lemma 4.3 (set $m = m_i = a(2D)^i \log n$ and $\tau = 1$), the probability that more than $2a(2D)^i \log n/(2D)^{i-1} = 4aD \log n$ samples are among the $\lambda_i = \frac{n}{c}$ closest samples to o is less than $2e^{-2aD \log n/3} = \frac{1}{n^{\Omega(a)}}$. The proof of Property 3 is identical, except that we replace λ_i by λ_{i-1}. Then, we have $c = (2D)^{i-2}$, $\frac{2m_{i+1}}{c} = 16aD^3 \log n$, and the probability that $|S_{i+1} \cap \beta_o(\lambda_{i-1})| > 16aD^3 \log n$ is smaller than $\frac{1}{n^{\Omega(a)}}$, as before. For Property 4, we expect $8aD \log n$ objects to be sampled at level i among the $4\lambda_i$ closest objects to o. Again, by lemma 4.3, the probability that less than half that many objects get sampled is at most $\frac{1}{n^{\Omega(a)}}$. Finally, the proof of Property 5 is almost identical to the proof of Property 2. By choosing a large enough, we can make sure that the five

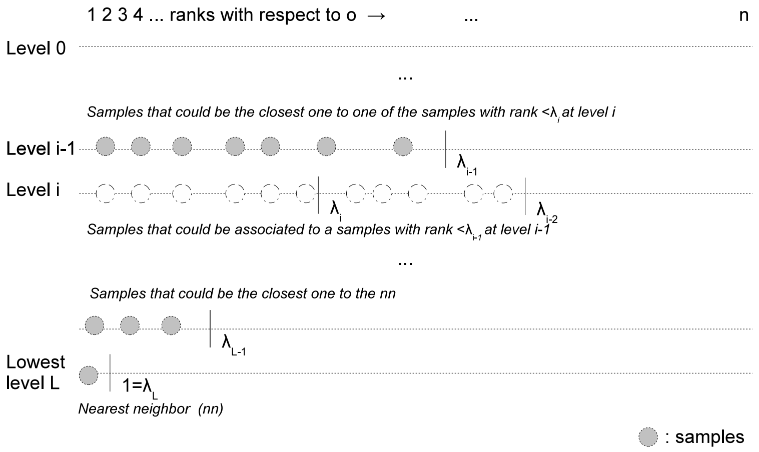

1 2 3 4 ... ranks with respect to o →
...
n
Level 0
...
Samples that could be the closest one to one of the samples with rank <λ_i at level i
Level i-1
λ_{i-1}
Level i
λ_i
λ_{i-2}
Samples that could be associated to a samples with rank <λ_{i-1} at level i-1
...
Samples that could be the closest one to the nn
λ_{L-1}
Lowest level L
1=λ_L
Nearest neighbor (nn)
: samples

Figure 4.1: We place all objects on the line such that the object u with rank $r_o(u, \mathcal{T}) = r$ is located at distance r from o.

properties are true for all objects and all levels w.h.p. (take the union bound over the n objects and the $L = \frac{\log n}{\log 2D}$ levels).

From now on, we assume that we are in the situation where Properties (1) to (5) are true for all objects (which is the case w.h.p.). Again, fix an object o. Consider a sample $s \in S_{i+1}$ such that $r_o(s) \le \lambda_{i+1}$ (note that Property 1 guarantees that there is a least one such sample). Further, let $s' \in S_i$ be the sample at level i closest to s *i.e.,* $s' = \min_{x \in S_i} r_s(s')$. Again, by Property 1, we know that $r_s(s') \le \lambda_{i+1}$. Hence, by the approximate triangle inequality 3 (see Section 4.2), we have:

$$r_o(s, \mathcal{T}) \le \lambda_{i+1} \text{ and } r_s(s'), \mathcal{T}) \le \lambda_{i+1} \quad \Rightarrow r_o(s'), \mathcal{T}) \le 2D\lambda_{i+1} = \lambda_i$$

Consequently, we know that the sample that is closest to o at level $i+1$ will be in the bin of a sample $s' \in S_i$ that has rank $r_o(s', \mathcal{T}) \le \lambda_i$. The algorithm associates every object o to the closest sample on each level. Hence, to find that sample for object o at level $i+1$, it would be sufficient to rank (w.r.t. o) all sample in S_{i+1} that fell in the bin of a sample at level i that has rank less than λ_i. Property 2 tells us that $|S_i \cap \beta_o(\lambda_i)| \le 4aD \log n$. Hence, by inspecting the bins of the at most $4aD \log n$ closest samples to o at level i, and ranking the samples at level $i+1$ that fall in those bins, we are guaranteed to find the closest sample (that is what we do on Line 3.7 of Algorithm 3). Property 4 tells us that all of the $4aD \log n$ closest samples to o at level i have rank less than $8\lambda_i$. Consider a sample $s \in S_i$ such that $r_o(s, \mathcal{T}) \le 8\lambda_i$ and a sample $s'' \in S_{i+1}$ that falls in the bin of s. By property 1, we must have $r_{s''}(s, \mathcal{T}) \le \lambda i + 1$. Thus, by inequality 2, we have:

$$r_{s''}(s, \mathcal{T}) \le \lambda_{i+1} \text{ and } r_o(s, \mathcal{T}) \le 8\lambda_i \Rightarrow r_o(s'', \mathcal{T}) < D(8\lambda_i + \lambda_{i+1}) \le 4\lambda_{i-1}$$

By property 5, there are at most $O(D^3 \log n)$ such samples at level $i+1$. This is the maximum cardinality of $c_j(i)$ on line 3.7 of the Algorithm 3.

To summarize, at every level in the hierarchy, and for every object, we need to rank at most $O(D^3 \log n)$ samples (which is what we do on Line 3.12 of Algorithm 3). Consequently, we need to ask at most $O(nD^3 \log^2 n \log\log n^{D^3})$ questions in total to rank at most $O(D^3 \log(n))$ objects for every object and level. The algorithm only fails with negligible (smaller than $\frac{1}{n}$) probability if an object has no sample that falls within λ_i at any level i. □

The proof of Theorem 4.1 is then immediate.

Proof of Theorem 4.1. The upper bound on the number of questions to be asked in the learning phase is immediate from Theorem 4.4. For each object, we need to store one identifier (the identifier of the closest object) at every level i in the hierarchy, and one bit to mark it as a member of S_i or not. Hence, the total memory requirements[6] do not exceed $O(n \log^2 n / \log(2D))$ bits. Finally, the properties 1-5 shown in the proof of Theorem 4.4 are also true for an external query object q. Hence, to find the closest object to q on every level, we need to ask at most $O(D^3 \log^2 n \log\log n^{D^3})$ questions. In particular, the closest object at level $L = \log_{2D}(n)$ will be q's nearest neighbor w.h.p. □

Note that this scheme can be easily modified for R-nearest neighbor search. At the i^{th} level of the hierarchy, the closest sample to q will, w.h.p., be one of its $\frac{n}{(2D)^i}$ nearest neighbors. If we are only interested in the level of precision, we can consequently stop the construction of the hierarchy at the desired level.

4.4.2 Lower Bound

In this section, we first construct a configuration of n objects, and define a distribution over query points, such that the disorder constant is D. Then, we show that no search algorithm can be guaranteed to find the nearest neighbor of the query point in less than expected $\Omega(D \log \frac{n}{D^2} + D^2)$ questions. We make the assumptions that all possible questions related to the n database objects can be asked during the learning phase, and even that the structure of the database is known. Then, we attach a query point to the database constellation in a random way. Consider the graph shown in Fig. 4.2. It is a star with α branches, each composed of n/α^2 *supernodes.* All edges in this part of the graph have weight 1. Inside each supernode, there are α database objects. A *root* node that connects the supernode to the other supernodes, and α objects, each connected to the root with a different edge. The weights of these edges range from $1/4\alpha$ to $\alpha/4\alpha$. Finally, the query point will be connected to one object on every branch of the star. Hence, the query point has α *direct neighbors* (one on each branch of the star). The edges connecting the query point to the graph have weights ranging from 1 to $1+\epsilon$, where $\epsilon \ll 1/4\alpha$. Note that we cannot know which are the direct neighbors of the query point, nor what the weights

---

[6] Making the assumption that every object can be uniquely identified with $\log n$ bits.

of the corresponding edges are. Thus, given the n database objects and the answers to all possible questions we can ask about the database, we need to find the nearest neighbor of the query point. First, we show that this structure has disorder $\Theta(\alpha)$.

Lemma 4.4. *The graph shown in Fig. 4.2 with the shortest path distance has disorder constant* $D = \Theta(\alpha)$.

Proof. Consider the configuration given in Figure 4.2. We need to show that for all triples x, y, z, where $x, y, z \in \mathcal{T} \cup \{q\}$, we have $r_x(y) \leq D(r_z(x) + r_z(y))$. First, let us consider two nodes x and y such that $d(x, y) = d$, with $d > 1$. Clearly, these two nodes must be in two different supernodes as the maximum distance inside a supernode is is strictly smaller than 1. Further, we have $|\beta_x(d)| \leq 4\alpha^2 d$, $\forall x \in \mathcal{T} \cup \{q\}$. Indeed, even if x is in the supernode at the center of the star, there are at most αd other supernodes within distance d. Each supernode can contain at most α nodes. Further, the query point could be within distance d of x, in that case there could be at most $2d\alpha^2$ additional objects in the balls. On the other hand, we have $|\beta_z(d/2)| \geq d\alpha/2$, $\forall z \in \mathcal{T} \cup \{q\}$. Indeed, even if z is placed at the end of a branch, there are at least $d\alpha$ supernodes within distance d, each containing α nodes. Hence, we have $r_x(y) \leq 4\alpha^2 d \leq 2D\alpha d \leq D(r_z(x) + r_z(y))$ by setting $\alpha = D/2$. We have used the fact that $r_z(x) + r_z(y) \geq |\beta_z(j)| + |\beta_z(d - j)| \geq 2|\beta_z(d/2)|$.

If the distance is smaller than 1, then x and y must be inside the same supernode. In that case, we have $r_x(y) \leq \alpha \leq 2D \leq D(r_z(x) + r_z(y))$. We can prove the other inequalities in a similar way. ☐

In the proof of Theorem 4.2, we show that we can lower bound the expected running time of any randomized algorithm on the example of Figure 4.2. The idea of the proof is that we must identify and compare all direct neighbors to the query point, and then find the nearest neighbor among the direct neighbors. We show that we cannot identify all direct neighbors in fewer than an expected $\Omega(D \log \frac{n}{D^2} + D^2)$ questions.

Proof of Theorem 4.2. Consider the graph metric with shortest path distance in Figure 4.2. Yao's minimax principle (see [MR95]) states that, for any distribution on the inputs the expected cost for the best deterministic algorithm provides a lower bound on the expected running time of any randomized algorithm. The graph structure (solid lines in Figure 4.2) is known, but the weights of the edges are unknown. The graph consists of a star with α branches, each composed of $\frac{n}{\alpha^2}$ supernodes. Each of the supernodes in turn contains α database objects (*i.e.,* objects in $\mathcal{T}$). Clearly, in total there are $\alpha\alpha\frac{n}{\alpha^2} = n$ objects. We know the answers to all questions of the type $\mathcal{O}(a, b, c)$, where $a, b, c \in \mathcal{T}$. We attach a query point q to that graph, and we assume that each "position" of the query point (as shown in Fig. 4.2) is equally likely. That is, the query point is attached to one (non-root) object chosen u.a.r. on each branch of the star with an edge. This object is a called a *direct neighbor.* The weights of the corresponding edges are chosen between 1 and $1 + \epsilon$ in a random way as well

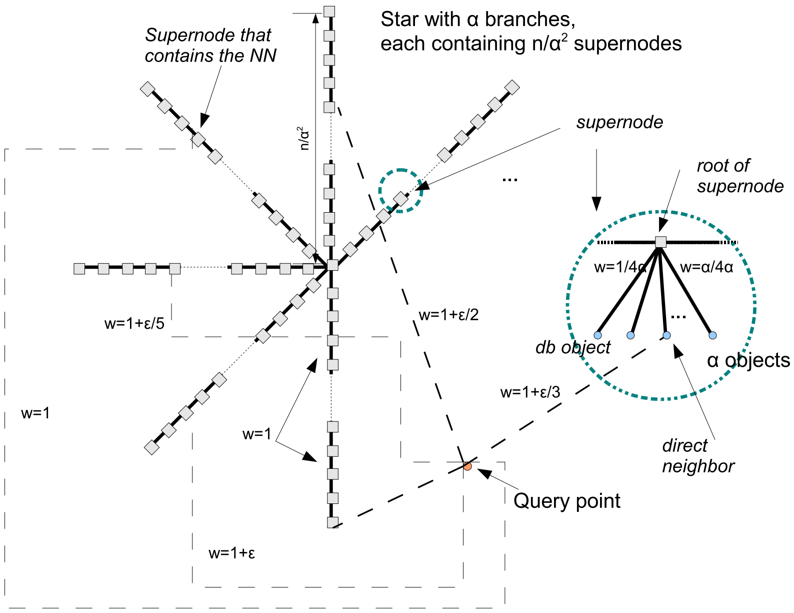


Figure 4.2: A graph with disorder constant α and shortest path distance. The graph forms a star with α branches. Each branch is composed of n/α^2 "supernodes". Each edge between the "roots" of the supernodes in the star has weight 1 (w denotes the weight in the figure). Each supernode (see zoomed region on the right side of the figure) is in turn composed of a "root", and a smaller (α)-ary tree (of depth 1) consisting of α actual database objects. The weights of the edges in the tree range from $1/4\alpha$ to $\alpha/4\alpha$ (are picked u.a.r. in this range). Finally, a query point is randomly connected to *one* non-root node on *each* branch of the star with edges of weights ranging from 1 to $1+\epsilon$ (dashed lines), where $\epsilon \ll 1/4\alpha$. The distances on the graph are shortest path distances. There are $(\frac{n}{\alpha})^{\alpha}$ ways to connect the query point to the database. Further, for each such *configuration*, there are α possible choices for the nearest neighbor (the direct neighbor which is connected to the query point with the edge of smallest weight). We assign weights to the edges connecting direct neighbors to the query point in such a way that each of the direct neighbors is equally likely to be the nearest neighbor, and each weight is different.

(such that we do not have ties, and each of the direct neighbors is equally likely to be the nearest neighbor). In other words, the input distribution is uniform over all configurations. First, note that q's nearest neighbor must be one of the objects connected directly to it *i.e.,* one of the α direct neighbors. Indeed, let $\delta = \{u \in \mathcal{T} | u \text{ is a direct neighbor of } q\}$. Then, we have $d(u,q) < d(v,q)$, when $u \in \delta$ and $v \in \mathcal{T} \backslash \delta$. Further, any of these direct neighbors could be q's nearest neighbor with equal probability.

Assume that we are given for free the answers to all questions, except the questions of the type $\mathcal{O}(q,x,y)$, where both $x, y \in \delta$. This amounts to knowing which are the direct neighbors (*i.e.,* we know the set of α nearest neighbors to q), but not knowing anything about the ranking of those direct neighbors with respect to q. Indeed, by construction, all direct neighbors are closer to the query point than any other object in the database. Hence, if we used the oracle to compare a direct neighbor with another object (which is not a direct neighbor), the oracle would always answer that the direct neighbor is closer to the query point. So, we could not exclude one of the direct neighbors as the nearest neighbor (we do not learn anything about the nearest neighbor). Further, the comparison of two objects $z, t \notin \delta$ does not tell us anything, since we do not know the weight of the edges. For instance, the fact that the second closest object to q inside a supernode Φ containing a direct neighbor is closer than the second closest object to q inside another supernode Φ' also containing a direct neighbor does not tell us that the direct neighbor inside Φ is closer than the direct neighbor inside Φ'. Hence, in order to identify the nearest neighbor, the best deterministic algorithm must at least ask α questions to find the nearest neighbor among the direct neighbors (we must traverse the list of direct neighbors and ask the oracle to compare every object with the current best candidate). Consequently, we must first identify all direct neighbors, and then compare them with each other.

Note that there are $(\frac{n}{\alpha})^\alpha$ ways to choose the direct neighbors, and that each configuration is equally likely. Identifying all direct neighbors is equivalent to knowing which of these configurations we are in. Let X denote the random variable of which each outcome corresponds to a configuration. Then, the entropy of X is $\log(\frac{n}{\alpha})^\alpha = \alpha \log(n/\alpha^2) + \alpha \log \alpha$ bits. The answer to every question we ask the oracle will reduce the uncertainty about which configuration we are in. In order for the probability of error p_e to be equal to zero *i.e.,* in order to be sure that we found the all direct neighbors, Fano's inequality (see [CT06], p39) tells us that we must know at least a set of answers A such that $H(X|A) = 1$ bit to have the $p_e \geq 0$.

For every branch of the star, choosing a direct neighbor u.a.r. is equivalent to choosing a supernode u.a.r., and then a direct neighbor inside that supernode u.a.r. First, assume that we *know*, on each branch, in which supernode the direct neighbor is located. Let us focus on one branch, and the supernode on this branch containing a direct neighbor. Denote that supernode by Φ. In that case, in order to identify the direct neighbor in Φ, we must ask questions of the type $\mathcal{O}(q,a,b)$, where $a, b \in \Phi$. Asking a question where either a, b or both are outside Φ does not tell us anything about which object is the direct neighbor, as all objects inside Φ are closer to q than any object outside that supernode. Further, note that the answer to any question of the type $\mathcal{O}(q, a, b)$, where $a, b \in \Phi$ and $d(z, b) > d(z, a)$ is b only if b is q's direct neighbor in Φ. Hence, the answer to a question of this type allows us to exclude only one object at a time[7]. Hence, for each of the α supernodes that contain a direct neighbor to q, we must ask an expected $\Omega(\alpha)$ questions to identify the direct neighbor. Knowing all the direct neighbors, when the supernodes in which they are located are known, reduces the entropy by $\alpha \log(\alpha)$ bits. Indeed, there are α such supernodes, and α choices for the direct neighbor inside each of these supernodes (*i.e.,* if we fix the supernodes containing the direct neighbors, there are α^α ways to choose the direct neighbors). As every question only excludes one object inside a supernode as direct neighbor, in total we must ask $\Omega(\alpha^2)$ questions to the oracle.

Let us now remove the assumption that we know which supernodes contain a direct neighbor. There are $(\frac{n}{\alpha^2})^\alpha$ ways to choose the supernodes that contain the direct neighbors. The entropy for this random choice is consequently $\alpha \log(n/\alpha^2)$ bits. Thus, at best, we need to ask $\alpha \log(n/\alpha^2)$ questions (in the best case each question reduces the number of possible configurations by 2) in order to know in which supernodes the direct neighbors are located. In total, we consequently need to ask at least an expected $\Omega(\alpha \log \frac{n}{\alpha^2} + \alpha^2)$ questions, to reduce the entropy by $\log(\frac{n}{\alpha})^\alpha = \alpha \log(n/\alpha^2) + \alpha \log \alpha$ bits and having $p_e \geq 0$. By letting $\alpha = \Theta(D)$, we obtain the claim. □

4.5 Searching with Unknown Characterization

When we cannot use any characterization of the hidden space as an input, we cannot be sure to retrieve the nearest neighbor of a query point q, unless we ask $O(n)$ questions. Indeed, unless we go sequentially through all objects and compare them to the current nearest neighbor, there could always be an object closer to the query point. In Section 4.4, we heavily relied on the fact that by sampling objects u.a.r., we could find objects close to the query point. Knowing the disorder constant then allowed us to exclude other objects as near neighbors *i.e.,* we could be almost sure that some objects were not among the closest neighbors of the query point. More precisely, as illustrated in Section 4.5.1, knowing the disorder and partial rank information, we can easily limit ranks. If we do not know D or another characterization, we can still hope that by building a hierarchical decomposition, dissimilar objects will be separated rapidly as we walk down from the root to a leaf. Hence, we would also expect objects similar to q to be close to it in the tree. However, we cannot bound this rank relationship, as we cannot use a characterization as an input to the algorithm. First, however, we will give an example of a simple and intuitive

---

[7] The same is true if we ask questions where a and b are in different supernodes. What matters is that we can only exclude one object as being a direct neighbor every time we ask a question.

algorithm that shows how much we can gain by knowing the disorder constant, or equivalently what we lose if we do not have such a characterization.

4.5.1 Consequences of knowing the disorder constant

If we know that an object u is the j^{th} nearest neighbor of an object x (*i.e.,* we have $r_x(u, \mathcal{T}) = j$), and we are looking for an object y, such that $r_u(y, \mathcal{T}) < \zeta$, then we know that y must lie in an annulus centered at x of a certain width around u *i.e.,* we know that $a \leq r_x(y) \leq b$, where a and b are functions of j and ζ. In particular, we have:

Lemma 4.5. *Consider three objects x, y, and u. Let $r_x(u) = j$ and $r_y(u) < \gamma$ (1), or $r_u(y) < \zeta$ (2). Then, y must lie in an annulus such that $\frac{j}{D} - \zeta \leq r_x(y) \leq D(j + \zeta)$.*

Proof. The result follows directly from the approximate triangle inequality (see Definition 4.2). The lower bound follows from inequality 3 for (1) and inequality 2 for (2). The upper bound follows from inequality 2 for (1) and inequality 3 for (2). ☐

By sampling m objects u.a.r., and computing all ranks w.r.t. to these objects, we can thus narrow down the search space to an annulus of width depending on D and on the rank of the closest sample.

Theorem 4.5. *Given a query object $q \in \mathcal{K}$, we can retrieve one of its R nearest-neighbors in $\mathcal{T}$ by asking an expected $m + \log n + D + \frac{D^2 n}{mR} + 1$ questions, with constant probability. The learning phase requires asking an expected $O(nm \log n)$ questions to sort all objects w.r.t. the m samples.*

Proof. During a learning phase we sample m objects $S = \{s_1, ...s_m\}$ u.a.r in $\mathcal{T}$, and rank all other objects with respect to the objects in S *i.e.,* $\forall s \in S, u \in \mathcal{T}$, we compute $r_s(u, \mathcal{T})$ by querying the oracle (this can be done by asking $O(mn \log n)$ questions). In the search phase, we start by finding the point in S closest to q, that is we want to find $x = argmin_{s \in S} r_q(s)$. This can be done in m steps by traversing the list of objects in S sequentially and storing the closest element seen so far. In particular, for every object in S, we ask the oracle whether it is closer to q than the current minimum, and if so it becomes the new minimum. Then, using binary search, we can find $j' = r_x(q)$ (*i.e.,* we ask the oracle whether q is closer to x than the element y such that $r_x(y) = n/2$, and then apply this process recursively on the new "interval"). Now, given that in the learning phase we sample m objects u.a.r. in $\mathcal{T}$, we know that $\mathcal{P}[r_q(x) = j] = \frac{m}{n}(1 - \frac{j}{n})^m$ (j has the distribution of the minimum of m integers selected uniformly at random between 1 and n). Further, we know by triangle inequality that $\frac{j}{D} \leq j' \leq Dj$. Hence, by Lemma 4.5, all objects o such that $r_q(o, \mathcal{T}) < R$ must lie in an annulus centered at x such that $\frac{j}{D^2} - R \leq r_x(o) \leq D^2 j + DR$ (see Figure 4.3). This annulus contains at most $(D+1)R + j(D^2 - \frac{1}{D^2}) < (D+1)R + j(D^2)$ objects, of which R are the R nearest

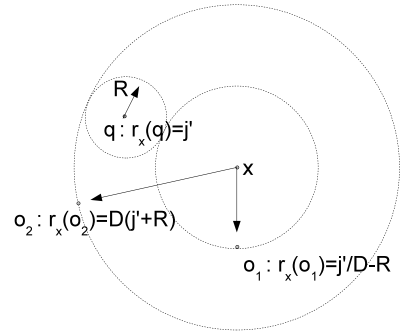


Figure 4.3: The R nearest neighbors of q must lie in an annulus around x.

neighbors of q. Hence, by sampling $(D+1)+\frac{j(D^2)}{R}$ times, we will retrieve an R-nearest neighbor with constant probability. Thus, the expected number of times we need to sample is $\sum_{j=1}^{n}\frac{m}{n}(1-\frac{j}{n})^m((D+1)+\frac{j(D^2)}{R}) \leq (D+1)+\frac{nD^2}{mR}$. For every sample, we ask the oracle if it is closer to q than the currently closest sampled point. If so, we store this point, else we delete it. ☐

In particular, by setting $m=\sqrt{\frac{n}{R}}$, we can retrieve one of the R nearest neighbors with constant probability in expected $O(D^2\sqrt{\frac{n}{R}})$ questions. The example is similar to what happens on a given level in the scheme of Section 4.4.1, where we applied this idea to develop a hierarchical scheme. In some sense, we applied this scheme repeatedly for smaller and smaller values of R, while reducing the search space. The fact that we know D, as is illustrated by the algorithm above, allows us to exclude some objects as being nearest neighbors. Indeed, if we have information about the rank of the sample w.r.t. the query point, or vice-versa, then we know the nearest neighbor must lie in an annulus of known width. On the other hand, if no characterization is known, we cannot exclude any object, whatever the density of the sampling. In the next section, we ask whether we can build a data structure that adapts to the characteristics of the space, without requiring them as input. In other words, we ask whether we can decompose the space in such a way that dissimilar objects are likely to be separated, and similar objects remain close to each other, without knowing a characterization of the space.

4.5.2 Binary Tree Decomposition

A natural and simple way to build a data structure suited for search operations is to build a tree. By recursively applying Algorithm 5 (see Figure 4.4), we can decompose the database into a binary tree. Clearly, this algorithm does not

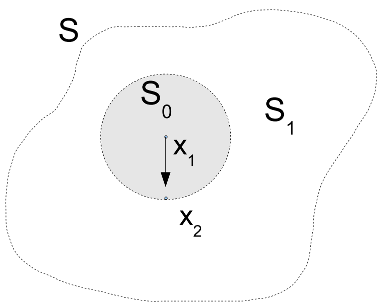


Figure 4.4: We decompose a set S into two sets S_0 and S_1. To do so, we select two points x_1 and x_2 u.a.r. in S, and let $S_0 = \{u \in S | r_{x_1}(u) < r_{x_1}(x_2)\}$ and $S_1 = \{u \in S | r_{x_1}(u) \geq r_{x_1}(x_2)\}$.

require any characterization of the space as an input. As illustrated in Section 4.5.1, if we do not know a characterization of the hidden space, we cannot limit the ranks if we only have partial rank information. However, we can expect this decomposition to adapt to the structure of the underlying space. In other words, even though we do not know any characterization of the space, we expect that if the space is sufficiently homogeneous similar nodes are more likely to end up in the same set than dissimilar objects. Let the expected diameter after the decomposition of S into S_1 and S_0 be defined as $\tilde{\Delta}_S = \frac{|S_0|}{|S|}\Delta_{S_0} + \frac{|S_1|}{|S|}\Delta_{S_1} \leq \Delta_S$ (by analogy to the notion in [DF08]). Observe that the diameter of a set (see definition in Section 4.2) has the following property.

```
input  : A set S of objects $\in \mathcal{T}$
output : Two sets of objects $S_0$ and $S_1 = S \setminus S_0$
pick two objects $x_1$ and $x_2$ u.a.r. in $S$;
$S_0 = \emptyset$, $S_1 = \emptyset$;
forall $u \in S$ do
    if $\mathcal{O}(x_1, x_2, u) = u$ then $S_0 = S_0 \cup u$ else $S_1 = S_1 \cup u$;
end
```

Algorithm 5: Rank-Ball Cut

Lemma 4.6. *The diameter of a set S with $|S| = n$ is always less than or equal to $2n$ i.e., $\Delta_S \leq 2n$, with equality when $d(u,v) = d(v,u)$ in the hidden metric space (symmetric distance function).*

Proof. Clearly, if $|S| = n$, for any pair of point u and v, we have $r_u(v, S) \leq n$. Hence, $r_u(v) + r_v(u) \leq 2n$ for all u, v. In case the distances are symmetric in the hidden space, we can rank the distances from the smallest distance to the largest distance. Consider the pair v, w, such that $d(v, w) = d(w, v) > d(i, j)$, for all i, j. Then, we clearly have $r_v(w) = n$ and $r_w(v) = n$, since there cannot be any point further away from v than w, and vice-versa. □

We can compute the expected diameter after the decomposition of S into S_0 and S_1. First, observe that it will always decrease, as the cardinality of the two new sets must be smaller or equal to the diameter of S. Let us denote by x_1 and x_2 the two randomly selected points in the set S. Let $r_{x_1}(x_2) = k$. By the approximate triangle inequality (1), for any pair of points u and v in S_0, we have $r_u(v) \leq D(r_{x_1}(u) + r_{x_1}(v)) \leq 2Dk$. Hence, the diameter Δ_{S_0} must be smaller or equal to $4Dk$. We can then easily compute the expected diameter to be $\tilde{\Delta}_S \leq \frac{4D}{n}k^2 + 2n - 2k$. Further, the optimal value for k is $k = \frac{n}{4D}$. However, by choosing x_2 at random, we cannot ensure that $r_{x_1}(x_2)$ takes a specific value. Nevertheless, we know that the value of k is uniformly distributed between 1 and n. Assume that we want $\tilde{\Delta}_S \leq \epsilon 2n$, for some $\epsilon < 1$. Then, we can prove the following theorem:

Theorem 4.6. *Let $\epsilon < 1$. Then,*

$$\mathcal{P}\left[\tilde{\Delta}_S \leq \epsilon 2n\right] = \frac{1}{2D}\sqrt{1 - 8D(1-\epsilon)}$$

Proof. We need to compute the probability that k is such that $\frac{4D}{n}k^2 + 2n - 2k < \epsilon 2n$, or equivalently $\frac{2D}{n}k^2 - k + (1-\epsilon)n < 0$. Solving for k, we obtain $k = \frac{1 \pm \sqrt{1-8D(1-\epsilon)}}{\frac{4D}{n}} = \frac{n}{4D} \pm \frac{n}{4D}\sqrt{1 - 8D(1-\epsilon)}$. Hence, the number of values of k for which the above condition is fulfilled is $|\frac{n}{4D} + \frac{n}{4D}\sqrt{1 - 8D(1-\epsilon)} - \frac{n}{4D} + \frac{n}{4D}\sqrt{1 - 8D(1-\epsilon)}| = \frac{n}{2D}\sqrt{1 - 8D(1-\epsilon)}$. As we choose k u.a.r. from n values, we have $\mathcal{P}\left[\tilde{\Delta}_S \leq \epsilon 2n\right] = \frac{1}{2D}\sqrt{1 - 8D(1-\epsilon)}$ □

Let a "good cut" be a cut such that the diameter is reduced by epsilon. The probability that we reduce the diameter y a factor ϵ degrades with increasing values of D. Hence, even though the disorder constant is not an input to the algorithm, the performance will depend on the disorder constant. For instance, if D were constant, then we would reduce the diameter by a constant with constant probability. In general, we roughly need $\frac{\log(1/c)}{\log(\epsilon)}$ good cuts to divide the diameter by a constant c. In any case,

Lemma 4.7. *The depth of the binary tree is $O(\log n)$ w.h.p.*

Proof. Let $\delta < 0.5$ be a constant independent of n, D. Consider a particular path in the binary tree from the root to a leaf. Let n_i denote the number if objects in the set at level i and k_i the rank of x_2 w.r.t x_1 (*i.e.*, $r_{x_1}(x_2)$) chosen at level i. Let $X_i = 1$ if $\delta n_i \leq k_i \leq (1-\delta)n_i$. As k_i is distributed u.a.r.

in $1, .., n_i$, we have $\mathcal{P}[X_i = 1] = 1 - 2\delta$. If $X_i = 1$, the number of objects is reduced by a factor at least $1 - \delta$ at this level *i.e.,* $\max\{|S_0|, |S_1|\} < (1-\delta)n_i$. As there are n objects in total, we can not reduce the number of objects by a factor $(1-\delta)$ more than $s = \frac{\log(n)}{\log(1/(1-\delta))}$ times. In m levels on the path, the expected number of times we expect X_i to be equal to 1 is $\mu = (1-2\delta)m$. If we set $m = \frac{2as}{(1-2\delta)}$, for some constant $a > 1$ we have

$$\mathcal{P}\left[\sum_{j=1}^{m} X_j < \mu/2 = as\right] < O(1/poly(n))$$

By the Chernoff bound. There are at most n paths (one per leaf). Taking the union bound over these paths, we obtain the claim. □

An interesting fact is that the probability that a node u falls in the good set *i.e.,* the ball around x_1 is given by $\phi_u = \mathcal{P}[u \in \beta_{x_1}(r_{x_1}(x_2))] = \frac{1}{n}\sum_j (1 - \frac{r_j(u)}{n})$. Hence, "outliers" are likely to be put in the same bin as other outliers, while similar objects are likely to be put in the same bin. For instance, an object y far away from all other objects, such that $\forall u \in \mathcal{T}$ we have $r_u(y) = n$, will hardly ever be put in the good set. Conversely, if there is a set of very popular nodes, which have a low rank w.r.t. all other objects, they will often end up in the good set. Consequently, this function can be used to estimate how "central", or popular an object is (analogous to the notion of 1-median in [Ind99]). $Y_i = \sum_j \mathbf{1}_{\{\text{node } i \text{ is in the good set}\}}$, where the sum goes over randomly selected hash functions and $\mathbf{1}_{\{\}}$ is the indicator function, will provide such an estimate. It also implies that outliers are more likely to be separated from other objects. In particular, if we computed k times the result of a randomly chosen hash function h, Y_u would be roughly equal to $k\phi_u$. By sorting the Y_i's, we obtain a ranking of the objects by popularity. In the next subsection, we will try to exploit this property to design a hashing scheme.

4.5.3 Rank-Sensitive Hashing

We have developed the intuition that by randomly cutting out balls in the hidden space, it is more likely that similar objects will stay together, and dissimilar objects be separated. This should be sufficient, if we can amplify this property, to allow us to efficiently search for similar objects by using an appropriately chosen hash function. Indeed, we will now show how we can use this technique to develop a rank sensitive hashing scheme. The rank distortion provides us a sufficient condition for the scheme to work. Our hash function h selects two objects u.a.r in $\mathcal{T}$ (say x_1 and x_2), and assigns values $h(u) \in \{0,1\}$ to all objects u as follows

$$h(u) = \begin{cases} 1 & \text{if } \mathcal{O}(x_1, x_2, u) = u \\ 0 & \text{if } \mathcal{O}(x_1, x_2, u) = x_2 \end{cases}$$

Note that computing h requires asking a single question per object, and that the algorithm does not require any characterization of the space as input. The function h is $(r, (1+\epsilon)r, 1-\frac{f(r)}{n^2}, 1-\frac{f((1+\epsilon)r)}{n^2\gamma})$-rank-sensitive. This is the result of Theorem 4.3

Proof of Theorem 4.3. First, we compute the probability that the hash function h is different for two objects u and q.

$$\begin{aligned} p &= \mathcal{P}\left[h(u) \neq h(q)\right] \\ &= \textstyle\sum_{i,j\in\mathcal{T}} \mathcal{P}\left[h(u) \neq h(q) | x_1 = i, x_2 = j\right] \mathcal{P}\left[x_1 = i, x_2 = j\right] \\ &= \textstyle\sum_{i,j\in\mathcal{T}} \mathbf{1}_{\left\{r_{x_1}(x_2)\in\left[r_{x_1}(q), r_{x_1}(u)\right]\right\}} \mathcal{P}\left[x_1 = i, x_2 = j\right] \\ &= \textstyle\sum_{i\in\mathcal{T}} \mathbb{E}\left[\mathbf{1}_{\left\{r_{x_1}(x_2)\in\left[r_{x_1}(q), r_{x_1}(u)\right]\right\}}\right] \mathcal{P}\left[x_1 = i\right] \\ &= \tfrac{1}{n} \textstyle\sum_i \mathcal{P}\left[r_{x_1}(x_2) \in \left[r_{x_1}(q), r_{x_1}(u)\right] | x_1 = i\right] \\ &= \tfrac{1}{n^2} \textstyle\sum_i |r_i(u) - r_i(q)| \\ &= \tfrac{1}{n^2} ||\rho_u - \rho_q||_1 \end{aligned}$$

Hence, we have

$$\mathcal{P}\left[h(u) = h(q) | r_q(u) \leq r\right] = 1 - \frac{1}{n^2} ||\rho_u - \rho_q||_1 \geq 1 - \frac{f(r)}{n^2}$$

and

$$\mathcal{P}\left[h(v) = h(q) | r_q(v) \geq (1+\epsilon)\right] = 1 - \frac{1}{n^2} ||\rho_v - \rho_q||_1 \leq 1 - \frac{f((1+\epsilon)r)}{n^2\gamma})$$

Where the two inequalities follow from Definition 4.3. □

A special case is when the function f is linear. Then, we obtain the following result.

Corollary 4.1. *We can retrieve one of the $(1+\epsilon)r$-nearest neighbors in $\mathcal{T}$ of a query point q, with constant probability, by asking $n^{O(\frac{\gamma}{\epsilon})}$ questions, where γ is the rank distortion of $\mathcal{T}$, when the rank distortion function is linear (*i.e.*, $f(r) = cr$).*

Proof. The proof is analogous to the proof for locality-sensitive hashing for binary vectors provided in [IM98] (a short introduction to locality-sensitive hashing is provided in Section A-3). More precisely, for an (r, R, p, P)-rank sensitive hashing scheme, retrieving one of the R nearest neighbor of a query point q will requires $O(n^\theta)$ evaluations of the hash function. θ is defined as $\frac{\log\frac{1}{p}}{\log\frac{p}{P}}$. It can be shown that $\theta \leq \frac{1}{\frac{1+\epsilon}{\gamma}-1} = O(\frac{\gamma}{1+\epsilon})$. Indeed, the probabilities p and P take the same form as if we hashed binary vectors of dimension n^2/c, and let $r' = r$, and $(1+\epsilon')r' = (1+\epsilon)r/\gamma$. Then, $\theta \leq \frac{1}{\epsilon'}$ (see [IM98]). □

Intuitively, one situation where f is roughly constant is when the underlying space is close to a line in $\Re^d$. Further, our numerics have shown that even for higher dimensions, when the underlying space is homogeneous (*e.g.,* points distributed u.a.r. in a unit box with wrap around distances), the function f is very steep for small values of $r_u(v)$ and then almost linear. An example is

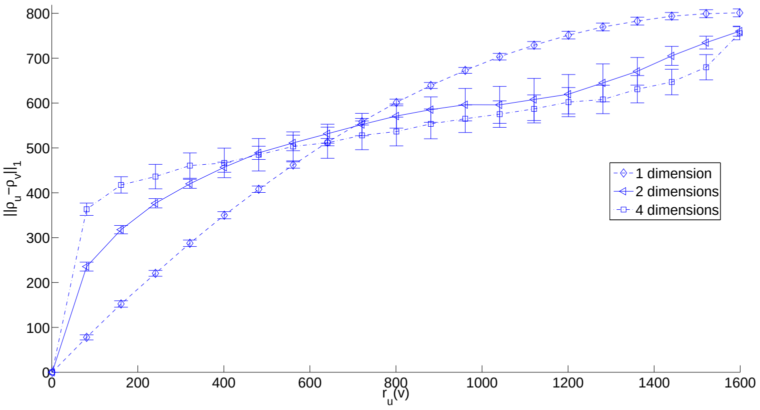


Figure 4.5: The hidden space consists of 1600 points distributed u.a.r. on $[0,1]^d$, where $d = 1, 2, 4$. To avoid border effects, we compute distances with wrap-around. We plot the $||\rho_u - \rho_v||$ against $r_u(v)$, for a fixed u. The results are averaged over 100 samples and the error bars correspond to the standard deviation. Note that the slope is first steep and then linear. Such a function is appropriate for RSH, as the function f increases monotonically. Further, the fact that we have a steep slope for small values of R make those spaces particularly attractive. Indeed, this implies that P decreases rapidly (so we can search for R-nearest neighbors, even for small R), and p is sufficiently large for small values of r. This example shows that for homogeneous spaces, the rank distortion function is such that we can perform RSH efficiently.

given in Figure 4.5. Note that in order for this scheme to allow us to retrieve the nearest neighbor of a query point q efficiently, it is sufficient to have a constant gap between $f(r_q(1))$ and $f(r_q(j))$, for $j \geq 2$. In the next subsection, we investigate what happens in high-dimensional Euclidean spaces, and show that such a gap exists as long as the query point q is sufficiently close to its nearest neighbor.

4.5.4 Rank-Sensitive Hashing in high-dimensional Euclidean Spaces

In order to illustrate and understand the behavior of the RSH algorithm of Section 4.5.3, we run simulations for normally distributed points in $\Re^d$. More precisely, in our setup we draw the positions of the n database objects according to $N(0, I_d)$, and choose a query point as follows. We choose one of the n points u.a.r, say x., and then select the query point u.a.r. in a box of side δ centered at x. For an (r, R, p, P)-rank sensitive hashing scheme (see definition 4.5), the number of questions we need to ask in order to retrieve one of the R nearest neighbors grows as n^ρ, where $\rho = \frac{-\ln p}{\ln(p/P)}$ (see [IM98], and Appendix A-3). In the case in point, we have no information about the distances in the underlying space. Hence, we can only fix a value for ρ (*i.e.,* fix the maximum number of questions we are willing to ask in the search phase), and "hope for the best". Our aim is to determine numerically, in this setup, the value of R we can expect for a fixed value of ρ as a function of the dimension of the space d.

To do so, for several values of the dimension d, we generated 100 point constellations (standard normal distributed) and placed the query point q as explained above (with $\delta = 0.1$). For each constellations and each point i, we estimated the probability that $h(i) = h(q)$, by sampling 1000 rsh functions. For a fixed ρ, we can compute $P = p^{\frac{1+\rho}{\rho}}$. In order to estimate R, we computed the distance ν between the query point and the furthest point j with $\mathcal{P}\left[h(j) = h(q)\right] \geq P$. Then, we estimated R as $R = |\mathcal{B}_q(\nu)|$. That is, for all points i outside $\mathcal{B}_q(\nu)$, we have $\mathcal{P}\left[h(i) = h(q)\right] < P$, and for the specified value of ρ, we can retrieve one of the R nearest neighbors.

Somewhat surprisingly, R decreases with increasing dimensionality. In particular, the probability that the query point and its nearest neighbor get the same hash value goes to one, while for all other points the probability that they get the same hash value as the query point remains bounded away from 1 by a constant. In Figure 4.6, we plot R as a function of the logarithm base 2 of the dimension. It can be seen that for large values of d, R tends to 1. This phenomenon is remarkable, as generally high dimensionality makes search more difficult. The distance oracle seems to allow us to actually search more efficiently in high dimensional spaces than in low-dimensional spaces.

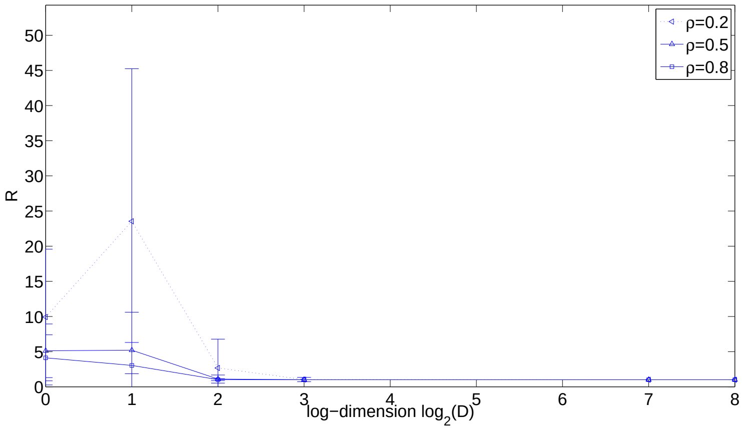


Figure 4.6: We plot R as a function of the dimension for 1000 points standard normally distributed in $\Re^d$. The query point is selected uniformly at random in a box of side 0.1 centered at a randomly selected point. The value of R decreases as the dimension increases. The results are averaged over 100 constellations of 1000 points. The error bars correspond to the standard deviation.

We now try and get an insight into why this is case. The following theorem has been shown by Demartines (see [Dem94], and also [FWV07]):

Theorem 4.7 (Demartines)**.** *Let $X \in \Re^d$ be a random vector with i.i.d. components: $X_i \sim \mathcal{F}$. Then,*

$$\begin{aligned} \mathbb{E}\left[||X||_2\right] &= \sqrt{ad-b} + O(1/d) \quad \text{and} \\ var\{||X||_2\} &= b + O(1/\sqrt{d}) \end{aligned}$$

where a and b are constants that do not depend on the dimension.

The theorem is valid whatever the distribution $\mathcal{F}$ of theX_i might be. Different distributions will lead to different values for a and b, but the asymptotic results remain. The theorem proves that the expectation of the euclidean norm of random vectors increases as the square root of the dimension, whereas its variance is constant and independent of the dimension. Therefore, when the dimension is large, the variance of the norm is very small compared with its expected value. Hence, for n points chosen independently and d large enough, if we apply Chebyshev's inequality and take the union bound over all n points, with an arbitrarily high probability all n norms will be $\Theta(\sqrt{d})$. Indeed, we have that $\mathcal{P}\left[|||X|| - \sqrt{ad-b}| > b\alpha\right] < \frac{1}{\alpha^2}$, for some constants a and b independent of d. By letting $\alpha = n$, where $c > 1$, we make sure that the bound holds for all points w.h.p.

We can now state the following theorem:

Theorem 4.8. *Let $\mathcal{T}$ be a set of n points in $\Re^d$, and let the position $x(i)$ of each object $i \in \mathcal{T}$ be chosen independently from $N(0, I_d)$. Let q be a query point. Let y denote its nearest neighbor in $\mathcal{T}$ and let $\min_{i \in \mathcal{T}} ||x(i) - q|| = ||y - q|| = \delta$. Then, (i) if $\delta = o(\sqrt{(d)})$, $\mathcal{P}\left[h(q) = h(y)\right] \to 1$, as $d \to \infty$, and (ii) if $\delta = \Theta(\sqrt{d})$, then $\mathcal{P}\left[h(q) = h(y)\right] \to \eta$, as $d \to \infty$, where $\eta < 1$ is a constant independent of d strictly smaller than* 1.

Proof. We have just shown (see Theorem 4.7) that all points lie very close to the surface of a d-dimensional sphere of radius $\Theta(\sqrt{d})$. By selecting two points x_1 and x_2 randomly for RSH, we hence roughly speaking select two points u.a.r on the surface of this sphere. Observe that for point v, w, t on the surface of the sphere, we have $||v-t||^2 > ||w-t||^2$ implies $v^T t < w^T t$. Consider a query point q at $x(q) \in \Re^d$ and a point $x(y) = x(q) + \Delta$, such that $||\Delta|| = \delta$. Let us denote by $F(q)$, $F(x_2)$ and $F(y)$ the projections of q, x_2 and y on x_1. Thus, $\mathcal{P}\left[h(q) \neq h(y)\right] = \mathcal{P}\left[F(x_2) \in [\,F(q), F(y)\,]\right]$. The normal distribution is a 2-stable distribution (see for instance [DIIM04]), and consequently $F(q) - F(y)$ is distributed as $X\delta$, where $X \sim N(0,1)$. Moreover, by the same argument, $x_2^T x_1$ has distribution $X'\Theta(\sqrt{d})$, where $X' \sim N(0,1)$.

Let us now consider case (i). The probability that q and y get a different value is the probability that $X'\Theta(\sqrt{d}) \sim N(0, \Theta(d))$ falls in an interval of width $Xo(\sqrt{d})$. Indeed, by Chebyshev's inequality, for d large enough, $Xo(\sqrt{d})$ will be $O(\sqrt{d})$ with an arbitrarily high probability. At the same time, the maximum of the the pdf of $N(0, \Theta(d))$ decreases as $\frac{1}{\sqrt{2\pi d}}$ with d. Hence, we have $\mathcal{P}\left[h(q) = h(y)\right] = 1 - \Theta(\frac{\delta}{\sqrt{d}}) \to 1$.

In case (ii), the probability goes to a constant. Indeed, with constant probability, $|X| > 1$ (about 0.32, according to the "68-95-99.7" rule). Hence, q and y are always at least a standard deviations of X' apart with constant probability. Further, the diameter of the d-dimensional sphere is $\Theta\sqrt{d}$, and the projection of x_2 on x_1 must lie inside this sphere. The probability that X' (which has standard deviation $\Theta\sqrt{d})$ falls in any interval of width $\Theta(\sqrt{d})$ inside the sphere is independent of the dimension. ☐

Hence, there is always a constant probability that an arbitrary point in $\mathcal{T}$ and q get separated. On the other hand, the probability that the query point and its neighbor get the same hash value goes to one for very large d, if the query point is sufficiently close (*i.e.*, within $o(\sqrt{d})$) of its nearest neighbor. Consequently, in very high dimensional spaces with normal distribution of points, the nearest neighbor can be retrieved in as little as $O(\log n)$ questions, as long as its distance to the query point is $o(\sqrt{d})$. Conceptually, in high dimensional spaces, all nodes will start looking "similar", except the nearest neighbor, which will lie relatively extremely close to the query point.

4.6 Concluding Remarks

We addressed the problem of finding an object similar to a query object among the objects in a large database. In contrast to most existing formulations, we asked whether the database can be searched efficiently if its distance information can only be accessed through a similarity oracle, and the underlying objects need not be in a metric space. The oracle is motivated by a human user who can make comparisons between objects but not assign meaningful numerical values to similarities between objects. This raises new interesting questions on what are good properties of the rank relationships, what are good and efficient algorithms and what is the right characterization of such a space. We worked with two such characterizations in this chapter. One that captures the transitivity of the rank relationship through disorder constant (D), and the other one, rank distortion (γ), which captures how rank $r_u(v)$ relates to $\mathbb{E}_i\left[r_i(v) - r_i(u)\right]$. We presented a new randomized algorithm that improves the performance of existing algorithms for the combinatorial framework, and proved a lower bound on the search complexity. We also propose a new characterization of the hidden space, *rank-distortion*, and show that the performance of a novel rank-sensitive hashing scheme depends on that property. Rank-sensitive hashing enables (approximate) nearest neighbor search in a manner similar to locality sensitive hashing. We believe that ideas of searching through comparisons form a bridge between many well known search techniques in metric spaces to perceptually important (non-metric spaces) situations. In Chapter 5, we present an architecture for a system that implements some of the ideas introduced in this chapter. Further, we present some experimental results for perceptual search. In particular, we asked human users to perform searches based on rank-sensitive hashing and evaluated the quality of the output for various amounts of training.

5

A Platform for Similarity Search

In this chapter, we present the architecture of an image database search engine based on a practical implementation of Rank-Sensitive Hashing (RSH), a notion introduced in Section 4.5.3 of Chapter 4. Our goal is to index images in such a way that one can find an image similar to a query image only by answering questions on relative similarity *i.e.,* only by making pairwise comparisons. Recall that our RSH function is used to map an image to a binary vector. Each coordinate corresponds to a ball in the “hidden” image space, and for a given image the corresponding bit is set to 1 if the image is inside that ball and 0 otherwise.

The scheme in Chapter 4 was based on the answers of a perfect similarity oracle. In practice, humans are not perfect oracles and consequently a few adjustments need to be made. In order to improve the performance of the system and reduce the number of questions we need to ask human users, we try and combine perceptual search based on answers given by users and automatic feature extraction based on image processing. The key idea is that we first use an image processing technique, in our case “eigenfaces” (see [TP91a, TP91b]), to compute approximate similarities between images. This might sound surprising, as we motivated the need for comparison-based search by the fact that image processing performs poorly. Further, we also argued that it’s extremely difficult to compute meaningful distances between images. Undoubtedly, image processing does not perform well. Notwithstanding, we can hope that this process helps us obtain a very rough idea of which images are similar, and most importantly helps us isolate the “hard” questions for which we use precious human interventions *i.e.,* the questions we need to ask the human users. In particular, pairs of very similar images or very different images are likely to be mapped to close by, respectively distant, positions. Thus, we intend to use human comparisons to refine the automatically generated similarity measures, for the triples of images we are the least confident about. The reference pairs of images (*i.e.,* the pairs used for RSH) are selected, as it is the case for RSH, u.a.r. among all possible pairs. Clearly, it seems impossible to design a system that will return the image most similar to a query image, among all the images in a database, for all users and queries. This is simply due to the fact that humans have different perceptions of similarity, and hence, even with infinite training, we could probably not pre-process the database such that efficient search is possible for all users. Nevertheless, we can try to speed up the search process, such that for most users the queried face will be one of the first answers provided by the system. At least, we can expect to design a system that performs substantially better than exhaustive search. Ideally, our system should present the images to the user sorted by relevance, analogously to what a web search engine does for websites. Hopefully, exhaustive searches will be extremely rare, and most searches will result in a rapid retrieval of the desired image or at least of a similar image. As mentioned above, in a real system, we do not have access to an exact distance oracle, which given a reference pair and a query image, returns the reference image closest to the query image. Humans will sometimes (or even often) disagree. If, for example, we fixed a reference pair (say images A and B) and a query image, and asked 100 humans to act as oracles, it is likely that some fraction f would answer A, and a fraction $1 - f$ would answer B. If f were close to 0 or close to 1, we could be pretty confident about the answer another random user would give to this question. On the other hand, if f were close to 0.5, A and B would probably both be as similar to the query image.

Likewise, our intuition is that an image processing technique such as "eigenfaces" will provide distances between images, but we need to decide which distances are meaningful. Recall that RSH consists in picking two images u.a.r., say A and B, and then to assign a bit 1 to an image C if it lies inside the ball $\mathcal{B}_A(d(A, B))$, and 0 otherwise. Clearly, if the distances provided by image processing corresponded exactly to human perception, we could use them to mimic the oracle. However, in practice we cannot make such an assumption. Nevertheless, we expect that if the distance $d(A, C) \ll d(A, B)$, or $d(A, C) \gg d(A, B)$, then we can trust the automatically generated distances and base the output of RSH on them. On the other hand, if those two distances are roughly the same, we need ask humans to reinforce this value.

We now first review the literature related to perceptual search and image similarity in Section 5.1. Then, we briefly explain the image processing technique that we use in Section 5.2. In Section 5.3, we present the architecture of an RSH implementation. Finally, in Section 5.4, we present our results.

5.1 Relationship to Published Works

Image processing and face recognition have been extensively investigated in the literature. In particular, two main problems related to ours were studied. First, the design of algorithms to extract features invariant to rotation, scaling, shifts, exposition, etc was investigated. Second, researcher tried to find meaningful ways to measure similarities, or distances, between feature vectors. By meaningful, we mean that the computed similarity must be related to the similarity perceived by humans. In this section, we review some of the publications related to those two problems. In [SR06b] and [SR06a], systems for perceptual navigation of an image database are presented. Roughly speaking, these systems provide a graphical user interface (GUI) for database navigation. Both systems are based on image processing. A map is displayed on which similar images (based on features extracted with image processing) are displayed together. In contrast to our approach, no effort is made to integrate human feedback in the training phase. More precisely, in [SR06b], images are displayed on a spherical visualization space that users can rotate. In [SR06a], a hierarchical navigation system based on multi-dimensional scaling (MDS, see [BG05]) is presented. In [Low04], a method is presented for extracting distinctive invariant features from images, that can be used to perform matching between different views of an object or scene. This method is well adapted if the same object, but rotated or shifted, must be retrieved from two different images. The authors of [KS04] present an extension of the approach in [Low04] that makes it more robust. They use principal component analysis instead of smooth histograms to the gradient patches. In [RTG00, Dem04, NSDW$^+$07, SJ99], various approaches to measure similarities between images are presented. In particular, in [RTG00], the authors propose to use the Earth Mover's distance to measure distances between feature vector. It is more robust than histogram matching techniques, in that it can operate on variable length representations of the distributions. In [NSDW$^+$07] and [SJ99], the authors introduce similarity measures based on fuzzy logic and fuzzy sets. Finally, Turk and Pentland [TP91a, TP91b], propose methods specific for face recognition and similarity measure. In this work, we apply their method, and we describe it in more details in Section 5.2.1. A comparative study of face recognition algorithms is presented in [HPAC03]. This study has the advantage that it includes an evaluation of image pre-processing techniques and of combinations of different techniques. Note that we based our choice of a face recognition technique on the results of this paper.

Another body of literature related to this work investigates issues related to *active learning*. The underlying assumption of active learning is that a large set of unlabeled data is available, but that asking users to assign labels to objects in order to train a classifier is costly. Learning theory asks whether it is possible to label only a subset of the objects in the database, and then infer the rest of the labels. In particular, the question arises of how to select that subset of examples to be labeled. The learning algorithm itself is allowed to select the subset of unlabeled examples to be labeled. However, in contrast with a *passive* learning algorithm, an active learning algorithm can select the objects to label iteratively. In particular, it can take into account the previously seen labels to select the labels to be revealed. For instance, for a linear classifier, an active learning algorithm would typically request the labels of the objects close to the current best guess for the decision boundary. In general, the approaches presented in the literature require that one can compute meaningful distances between objects before labeling takes place. In [DH08] for instance, the active learning algorithm takes a hierarchical decomposition of the database objects as input. Active learning relates to RSH, as in a certain sense users assign labels to pictures by making statements about similarity. We do not have to specify explicitly what the labels are, but we must select good reference images. Other references on active learning include [CGJ96, LG94, Set09, BHW08]. Though we do not explicitly use active learning techniques in the current approach, we believe that such techniques will be very useful to extend and improve our scheme. For instance, instead of choosing reference pairs for RSH u.a.r., we could try and select a new pair that is very discriminant.

5.2 Image Processing

Typically, images are stored as three $a \times b$ matrices R, G and B, where $P = ab$ is the number of pixels in the images. The matrices R, G and B represent the red, green and blue intensities of every pixel. Each entry in those matrices is an 8 bit vector[1], representing a value between 0 and 255. A pixel with intensities $(0, 0, 0)$ is completely black, while a pixel with intensities $(255, 255, 255)$ is completely white. As suggested by the authors of [HPAC03], who provide an extensive comparison of face processing technique, we pre-process images by resizing them if necessary, normalizing intensities and converting them to gray scale. Assume that a pixel i has intensities (r_i, g_i, b_i). Then, intensity normalizations consists in modifying, $\forall i$, the intensities as follows:

$$(r_i, g_i, b_i) \quad \leftarrow \left(\tfrac{r_i}{r_i+g_i+b_i}, \tfrac{g_i}{r_i+g_i+b_i}, \tfrac{b_i}{r_i+g_i+b_i}\right)$$

A its name indicates, this techniques aims at mitigating the effects of different brightnesses in different colors channels. Conversion to gray scale is simply done by replacing the RGB triple for every pixel by a single value, which is a weighted sum of the three color intensities. Different conversion formulas exist. A common conversion is as follows:

$$I_i = 0.3r_i + 0.59g_i + 0.11b_i$$

Note that the weights are based on experimental results. The gray scale image can now be stored as a single $a \times b$ matrix I. This image can then be converted to a vector Λ, simply by concatenating the rows of the matrix. In order to obtain a low-dimensional representation of the images, we project these vectors on so called eigenfaces. This technique is explained below. That is, we try to represent every image as a combination of "basis vectors" in the image space.

5.2.1 Eigenfaces

In this section we give a brief explanation of the eigenface method of face recognition, while referring the reader to Turk and Pentland [TP91a, TP91b] for

---

[1] the resolution could vary, but for the sake of simplicity, we make the assumption that there are 255 levels of intensity.

more detailed explanations. We chose this approach based on the comparative study in [HPAC03], and on the fact that when dealing with images of faces taken for large organizations, we can expect the image format to be fairly standard.

We compute the covariance matrix C, of facial images from a set of M *training* images in vector form $\{\Lambda_1, \Lambda_2, ..., \Lambda_M\}$ as follows:

$$\begin{aligned} \Psi &= \tfrac{1}{M}\textstyle\sum_{i=1}^{M} \Lambda_i \\ \Phi_i &= \Lambda_i - \Psi \\ A &= [\Phi_1 \Phi_2 ... \Phi_M] \\ C &= \tfrac{1}{M}\textstyle\sum_{i=1}^{M} \Phi_i \Phi_i^T \\ &= AA^T \end{aligned}$$

The eigenvectors and eigenvalues of this covariance matrix are calculated using standard linear methods and the M‘ eigenvectors with the highest eigenvalues chosen to formulate the projection matrix u. A face-key ω *i.e.,* an image vector projected into the face space, can then be produced by the following equation for an image Λ:

$$\omega_j \quad = u_j^T(\Lambda - \Psi) \text{ for } j = 1 \text{ to } M$$

These face-keys can than be compared using the Euclidean distance measure.

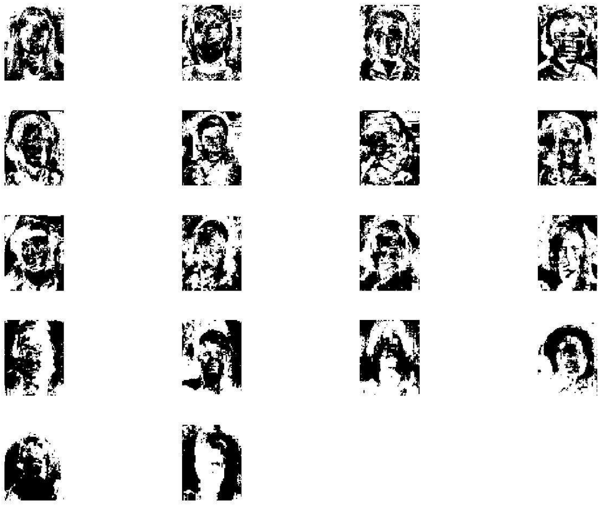


Figure 5.1: The 18 first eigenfaces for a set of 20 training images of size 82×65.

5.3 System

Assume that we are given a database $\mathcal{T}$ of n images. We intend to map each of these images to a RSH vector in $[0,1]^{k_n}$. Intuitively, k_n should be roughly $\lceil \log(n) \rceil$, so that a different binary vector can be assigned to every image. However, each of the entries corresponds to an "approximation" of the value of an RSH function as presented in Chapter 4, Section 4.5.3. Every coordinate should tell us, for a given reference pair A_j, B_j, if the image under consideration is inside or outside the ball $\mathcal{B}_{A_j}(d(A_j, B_j))$. It is an approximation, because neither image processing nor human training can provide us the exact value of the hash function. Rather, we will consider that a value larger to 0.5 indicates that the image is inside the ball, and a value smaller than 0.5 that it is outside. The closer the value is to 0.5, the less confident we are about it. This concept is illustrated in Figure 5.2.

The *learning* phase works as follows. For every image, as mentioned above, we intend to store a binary vector of length k_n, where each of the k_n entries corresponds to an approximation to the answer which the similarity oracle presented in Chapter 4 would give for a given reference pair of images. Hence, we choose k_n pairs of images u.a.r among the $\frac{n(n-1)}{2}$ possible distinct pairs to be the reference pairs for RSH. Obviously, the pairs are the same for all images and remain fixed once chosen. Every image is pre-processed when added to the database. First, an image I is resized, so that all images have the same size. Then, we normalize the intensity, as explained above in Section 5.2. Finally, the image is converted to grayscale, and its face-key is computed, as explained in Section 5.2.1. Then, for each of the k_n RSH reference pairs (A_j, B_j) and image I, we compute the ratio

$$r_{I,j} = \frac{||\omega_{A_j} - \omega_{B_j}||_2}{||\omega_{A_j} - \omega_{B_j}||_2 + ||\omega_{A_j} - \omega_I||_2}$$

where ω_x denotes the face key of image x (see Section 5.2.1). Clearly, if this ratio is close to 0, then the image I is outside the ball $\mathcal{B}_{A_j}(d(\omega_{A_j}, \omega_{B_j}))$, and if it is close to 1, we can be confident that it is inside. In Figure 5.3, we summarize this process. Human users come into play when a ratio is close to 0.5. For all triples for which this ratio $r_{I,j}$ is such that $\max\{r_{I,j}, 1 - r_{I,j}\} < \theta$, where $\theta \in [0.5, 1]$ is some threshold value, we can ask human users to refine it. In particular, consider a triple A_j, B_j, I, for which $r_{I,j}$ is close to 0.5. Assume that at some point in time, t users have acted as oracle for this triple, and a new user answers the question. Then, we simply recompute the ratio as

$$r_{I,j} \leftarrow \tfrac{t r_{I,j}}{t+w} + \tfrac{w}{t+w} r_{I,j}$$

and update $t \leftarrow t + w$. That is, the current value of the ratio is the average of the users opinions given so far. An arbitrary weight *e.g.,* a weight of 1, can be given to the initial ratio obtained with image processing, and an arbitrary weight of w to clicks. Straightforwardly, the higher w, the more importance we give to clicks.

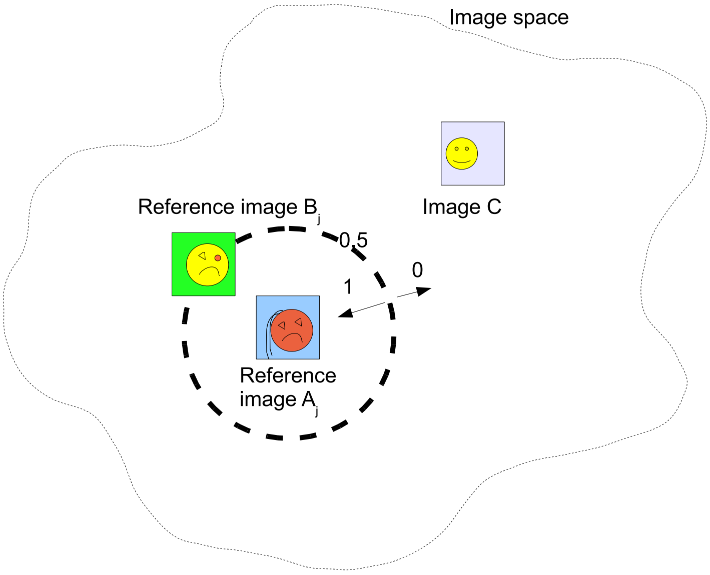


Figure 5.2: We illustrate how the j^{th} coordinate of the RSH vector for an image C is computed. The j^{th} coordinate corresponds to the reference pair A_j, B_j. If image C is inside the ball $\mathcal{B}_{A_j}(d(A_j, B_j))$, the hash value is 1, otherwise it is 0. However, neither image processing nor human training can tell us exactly where image C lies. Hence, we compute an initial value based on image processing, which lies between 0 and 1. The closer the value is to 0.5, the more uncertain we consider it to be. Later, this value will be refined through human training. If the image lies clearly outside the ball, most human should agree and training will push the value down to 0 (and vice-versa if it is clearly inside the ball).

In the search phase, we repeat the same process by asking the user the same k_n questions (corresponding to the k_n reference pairs), but for the query point. That is, for all of the k_n balls, we want to know whether the query point lies inside or outside. This time, we get a binary vector of answers $a \in \{0,1\}^{k_n}$. This is because in the search phase, a user will be shown every reference pair only once. Depending on which of the reference images the user clicks, a 1 or a 0 is stored. Finally, we sort the objects in the database according to $||a - r_I||_1$, $\forall I \in \mathcal{T}$, where r_I denotes the vector $(r_{I,1}, r_{I,2}, ..., r_{I,k_n})$. The user is presented the results from the most similar image to the least similar image. Potentially, one could improve the results by adding tags to images, and using them to break ties among images at close distances. Further, in order for the system to work well, different users should compare images roughly in the same way. In other words, comparisons should be based on the same criteria, and the

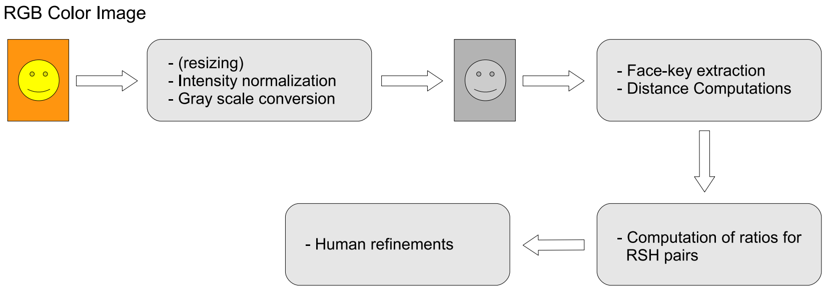
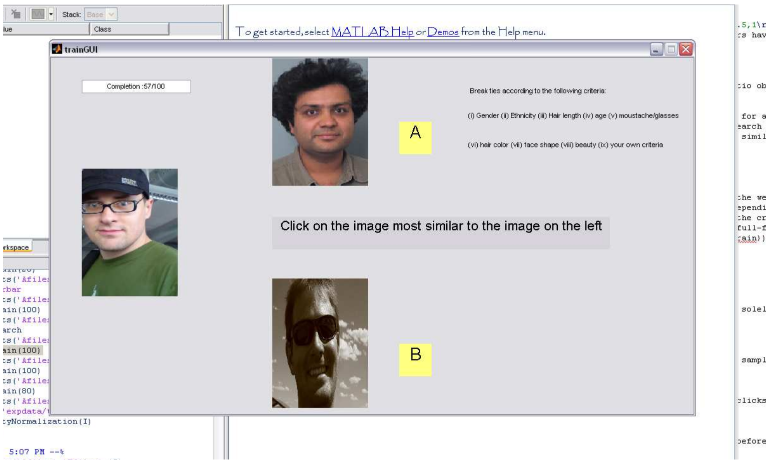


Figure 5.3: The block diagram for image insertion

weight given to each criteria should be the same. For instance, assume that the reference pair consists of an asian woman, and a Caucasian man, and that the query image is an asian man. Depending on the user, both reference images could be considered to be the most similar one to the query image. In order to mitigate this undesirable effect, we give the users an order on the criteria in our experimental setup, which they must use to break ties. For instance, gender always comes before ethnicity, and consequently the right answer above would be the caucasian man. We now present a few of these experimental results.

5.4 Experimental Results

We have implemented a full-fledged test system in Matlab, in order to obtain experimental results on perceptual search based on RSH (a screenshot of the training phase is shown in Figure 5.4). We tested perceptual search based on comparisons on a database of 110 faces, of different gender, ethnicity, age, etc. The images we used to test our implementation come from the Indian Face Database [JM02], a collection of faces of the University of Essex [Spa02], and a collection of pictures of friends and colleagues. We set $k_n = 10$, $\theta = 0.9$ and w, the weight of a click, to 10. The Eigenfaces were computed based on 20 training images. Seven users were each asked to search for two query images selected u.a.r in the database. This amounts to 20 clicks per user for searches. Additionally, they were asked to make 200 training clicks. We then computed the rank of the query image in the search results, after various amounts of training clicks. That is, we ranked the images with respect to their l_1 distance to the search vector as explained above, and looked at the rank of the query image in the result list. Note that the training clicks of all users were randomly mixed and shuffled. Clearly, if no image processing nor training were done, the average position of the query image in the result list would be in the middle, so roughly $n/2$. In Figure 5.5, we plot the average percentage of the database that



To get started, select MATLAB Help or Demos from the Help menu.

Completion :57/100

Break ties according to the following criteria:

(i) Gender (ii) Ethnicity (iii) Hair length (iv) age (v) moustache/glasses

(vi) hair color (vii) face shape (viii) beauty (ix) your own criteria

Click on the image most similar to the image on the left

Figure 5.4: A screen shot of the training phase in the Matlab implementation.

must be inspected before the query image is found (*i.e.,* the relative rank of the query image in the result list), as a function of the number of training clicks. It can be seen that the curve converges to an average value of roughly 20%. This means that on average the searched image (query image), will appear among the 22 first results presented to the user. Clearly, without any processing, this average would be close to 50% *i.e.,* 55 images. Image processing alone (without training) improves over exhaustive search by roughly 10%.

Collecting clicks from users is not necessarily an easy task, which explains the relatively small number of clicks and users. Nevertheless, the results in Figure 5.5 show that training has a clear impact on the quality of the search. Indeed, training improved the quality of the results by roughly 20% over perceptual search based uniquely on image processing. In itself, this result is interesting as it shows that different users agree more often than not. Indeed, training clicks from different users could have canceled each other out. After observing and talking to users, it appeared that most users find it rather simple to classify faces based on criteria such as ethnicity, age or gender. This fact certainly explains that training led to an improvment, as such distinctions are easy to make for humans, but difficult for computers. We also believe that this is how humans "compute" similarity, by tagging images and matching them whenever possible. On the other hand, when all faces are roughly similar and come from the same "group" of people, classification becomes much more difficult, and seems to be based on the face shape, and criteria specific to each user. Hence, the quality of the search results also depends on the composition of the

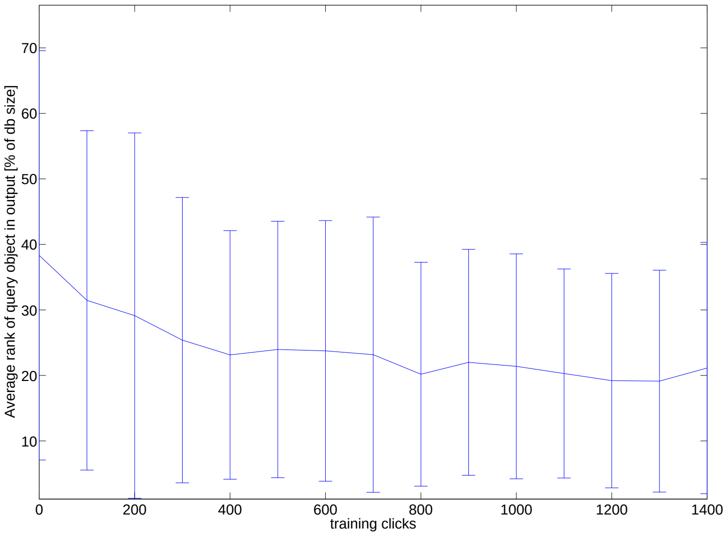


Figure 5.5: We show the effect of human training on RSH results. The plot shows the average percentage of the database that must be inspected before the query image is found, as a function of the number of clicks. The error bars correspond to the standard deviation

database. If the faces are very clustered in different group, search will be easier than if they all belong to the same group. Note also that in this experiment we tried to retrieve a specific image, and not any image that is similar to that image. This choice was motivated by the fact that we wanted to obtain sound numerical results. A closer look at the search results after all training clicks (see Figure 5.6) shows us that roughly 30% of the time, the searched image will be among the 10 first results (in our database of 110 images), and roughly 50% of the time among the 15 best results. On the other hand, 20% of the searches will produce very bad results *i.e.,* the searched image will not be among the 50 first images shown to the user. This result tends to indicate that there is some inherent slack in the process. In other words, for most of the images the RSH vector is meaningful and corresponds to some extent to human perception. On the other hand, for a part of the images, the process fails and they are very hard to retrieve.

5.4.1 Web Platform

In addition to the experimental results exposed above, we have implemented in the framework of several student project, a web platform for comparison based

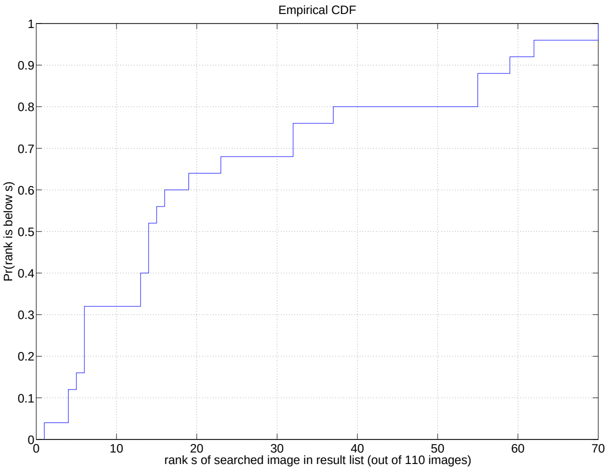


Figure 5.6: CDF of the quality of search results after training. It can be seen that roughly 50% of images appear among the 15 first search results. Conversly, there is a fraction of the images which are hard to classify and have rank higher than 50.

search. The current version of the website can be found at

http://ipg.epfl.ch/∼dtschopp/facebrowser/

. The underlying mechanisms and algorithms are the same as for the matlab implementation. Additionally, we offer the users the possibility to insert images, to tag them, and to add contact details for the person on the images. When new images are inserted, new reference pairs are automatically selected. Users can also navigate the database, either by moving from one image to one of the most similar images, or by moving to a completely different image (we added links to images with exponentially spaced ranks *e.g.,* to image with ranks $2, 4, 8, 16, 32, ...$ w.r.t. to the current image). The website is written in PHP, html and mySQL.

5.5 Concluding Remarks

In this Chapter, we have presented the architecture for a systems that implements image search based on rank sensitive hashing (RSH). The system allows users to retrieve an image in a database on average more rapidly than if they had to search the database exhaustively. In a certain sense, every pair of images selected to be part of the hashing scheme is a form of label, or coordinate. Every image in the database gets labeled automatically through image processing. Further, the labels we are most uncertain about are refined by human users. Those coordinates can then be used to compute distances between images.

Search is entirely based on a small sequence of questions, which is processed to output the label of the query image. In contrast to classical labeling schemes, we do not have to explicitly define labels. While these implementations are in early stages, we believe that this approach can be greatly improved. Our experiments have shown that human training can considerably improve the search results. Currently, we are very conservative in the training phase, as a training click only modifies one coordinate of one image. Though this can influence the outcome of a large number of searches, the impact remains small (an image can gain or lose one rank in the search results). Another possible option would be to use the training clicks to learn a distance function, and use this distance function to iteratively recompute the coordinates of the images. Most interestingly maybe, we could use active learning techniques to select the reference pairs for RSH. In particular, as the number of images in the database increases, we need to select new reference pairs. In that case, we could select those pairs based on the clicks collected so far instead of choosing them u.a.r. One possibility might be to select pairs that have not been well discriminated, and consequently lie in a region of the hidden space where objects are not well classifiable.

Conclusions and Future Work

6

In this thesis, we have discussed a routing and a search problem. The two problems share the particularity that we can exploit their low intrinsic complexity to develop efficient algorithm for the “retrieval” of a target node or database object. In both cases, the problem was selected because it seemed natural and intuitive that the problem could be solved if we managed to understand its structure. In turn, after characterizing the problem by an appropriately chosen property of the space in which the objects or nodes live, we could design good algorithms that depend only on this property. Indeed, it seemed natural to assume that the intrinsic dimension of the connectivity graph of very large wireless ad hoc networks was low dimensional. Clearly, this intuition comes from our perception of the physical world in which the nodes live. It appears natural that distant nodes have to communicate over mode relays than nearby nodes. It also seems natural that the topology, in general, changes more rapidly at the local scale than at the global scale. In the case of image databases, our intuition was that humans have the ability to classify pictures of faces based on a relatively small number of criteria. Hence, it must be possible to obtain a representation of a set of human faces that is low dimensional and easily searchable. Here, one of the main issues is that humans cannot assign meaningful distances to pairs of pictures, and that automatic comparison is difficult. Consequently, in addition to be low dimensional, sets of images should also be searchable by comparisons. In both cases, one of the difficulties is to identify and understand the right characterization of the problem.

Clearly, this work is mainly theoretical, and one possible continuation would be the design of a system to test and evaluate our results in a real world setup. In the case of image database search, and more generally search based on human perception, we believe that there is a lot of additional work that needs to be done in order to fully understand the problem. In particular, one could design more efficient hash functions for rank-sensitive hashing, and obtain more robust characterizations of the sets of images or objects in general. One option would be to combine automatic classification with human based search to obtain a new kind of search engine for images. Also, the majority of our results on this problem are based on the characterization of the hidden space given by the disorder constant. This criterion is a worst case criterion. Hence, on average, the results might be considerably better. We went in this direction by introducing the idea of rank distortion, which is an average criterion. However, it is not a trivial task to compute this distortion, even for simple spaces. Thus, we believe that it is still possible to make considerable progress on this problem. This work could potentially also be related to recent results on inference of rankings.

Regarding the routing problem, we believe that the doubling dimension is a robust notion of complexity. One interesting extension of our work would be to investigate trade-offs of control traffic overhead and throughput. That is, how much signaling traffic must be sent around in the network, in order to establish a path, or several paths with some guarantees on the throughput (*e.g.,* average or minimal throughput). Here, one could examine recent results on hierarchical cooperation (*e.g.,* groups of nodes can cooperate to mimic MIMO transmission schemes) in wireless networks, and try and design transmission schemes which implement such cooperation schemes. Another variation of the problem would consist in loosening the requirement that there must always exist all-to-all paths in the network. One could replace this requirement by the requirement that there must always exist a path for a fraction of the node pairs, or that a path must exist with a given probability. Potentially, it would be possible to design schemes that only guarantee the existence of a fraction of the paths, with a dramatic reduction of control traffic.

Appendix

A-1 Unit Disc Graphs

Another common model used in studies on wireless networks are *Unit Disk Graphs* (UDG), which are the deterministic variants of the random geometric graphs. The randomness of the positions of the nodes is removed and they can be placed arbitrarily on a finite of infinite area. The channel model is completely deterministic as before and nodes are connected if their Euclidean distance is below a threshold distance r, called the communication radius. In mathematical terms, two nodes u and v with positions $x(u), x(v) \in [0, R]^2$ are connected if and only if $||x(u) - x(v)|| < r$. We will now show that there exist UDG which are not α-doubling (see Section 3.2 for a definition of an α-doubling metric).

Theorem 6.1. *There exists an infinite UDG for which is no constant that upper bounds the doubling dimension* i.e., *UDG are not doubling.*

Proof. Consider the graph shown in Figure A-1. To show that this graph is not α-doubling, we must show that there exists no constant such that all balls of radius R can be covered a constant α number of balls of radius $R/2$, for all R. Consider the ball centered around u in the figure. One can see that there are $R/4 + 1$ "columns" which cross the middle row at a distance less than $R/2$ from u (that is, the intersection of the column and the row is less than $R/2$ hops away from u). The intersection of each of these columns with $B_u(R)$ is of length more than R (see hatched zones on Figure A-1). Consequently, for each of these columns there is at least one node at distance more than $R/2$ from the middle row. To cover these nodes, we need to place at least one ball of radius $R/2$ on each of these columns. Hence, the doubling dimension is lower bounded by $R/4$ and tends to infinity as R goes to infinity. □

One can notice that in the non-doubling UDG in the proof of Theorem 6.1 results from a careful construction. In Appendix A-2, we show however that such a structure will occur with high probability when $r_n < \sqrt{\log n}$ in random geometric graphs.

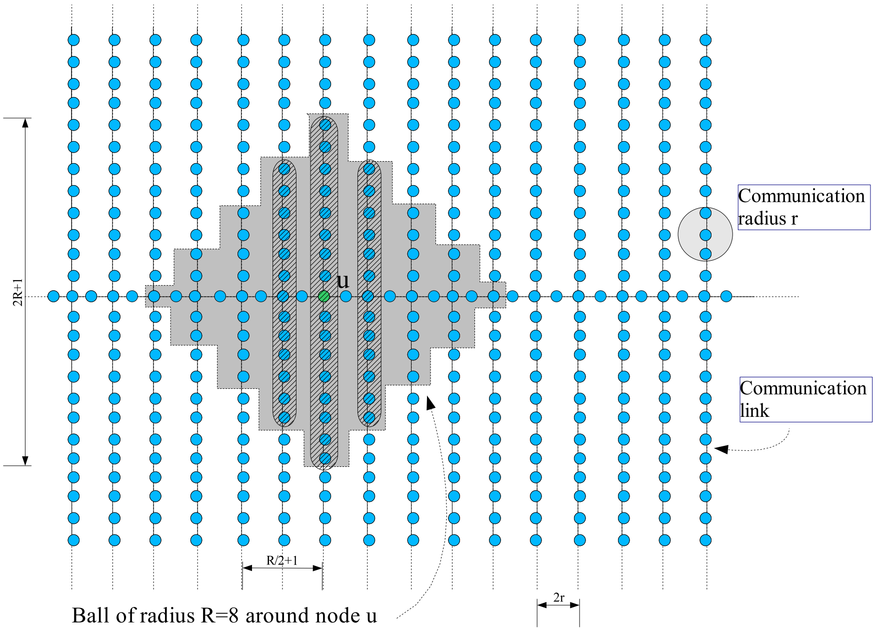


Figure A-1: An infinite UDG obtained by deleting all the nodes in every second column of a grid, except for the nodes on the the middle row. Consequently, "columns" are $2r$ apart.

A-2 Random Geometric Graphs with subthreshold Communication Radius

We first consider the case in which the communication radius r is such that $r_n = r = (\log n)^{\frac{1}{2}-\frac{\theta}{2}} < \log^{1/2} n$ and $\theta \in\ ]\zeta, 1\,]$. ζ is a constant such that $0 < \zeta < 1$.

Lemma 6.1. *For any constant β, there exists constants $\gamma > 0$ and $b > 0$ such that a small square area of side γr with b nodes contains a subgraph of doubling dimension $\beta + 1$ with probability $p > 0$.*

Proof. Consider the small square shown in Fig. A-2 of side γr, where γ is a constant independent of n to be specified later. Subdivide the small square further into mini-squares of side r/c. Choose the constant c such that there exists a constant k satisfying $\sqrt{2}(k-2) > c \geq \sqrt{k^2+1}$. Under these conditions, two nodes in mini-squares separated by $(k-2)$ other mini-square will be connected, but not mini-squares $r(k-2)\sqrt{2}/c$ apart (see right hand side of Fig. A-2). Consider now the graph on the left hand side of Fig. A-2. Assume that each full (colored) mini-square contains exactly one node. We now focus on the ball $\mathcal{B}^{\mathcal{G}}_{2R}(u)$ and will lower bound the number of balls of radius R necessary to cover it. On the $\lfloor R/2 \rfloor$ first vertical branches from the left, the last node

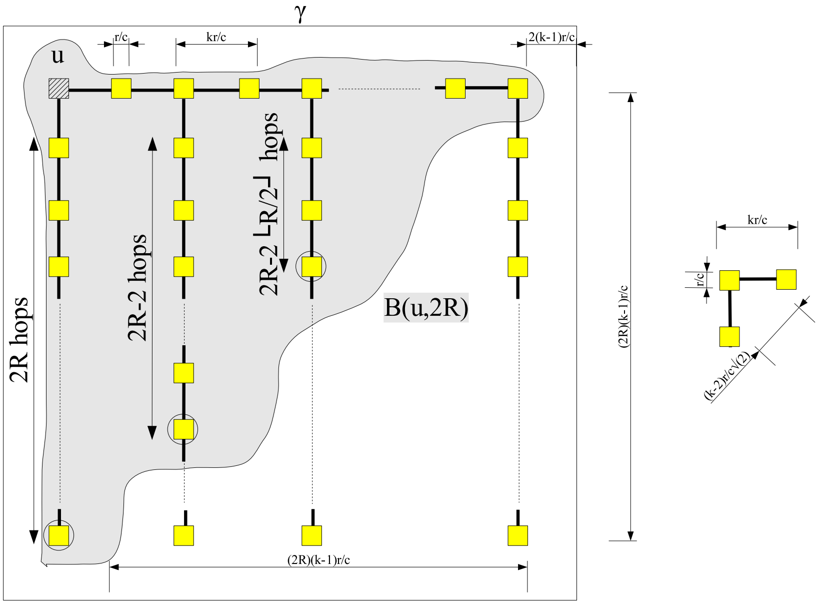


Figure A-2: A non-doubling subgraph in a random geometric graph

of the branch inside that ball (circled) must be covered by a ball of radius R centered on the same branch. This is clear since the length of the branch is larger than R. Consequently, the doubling dimension of this graph is at least $\lfloor R/2 \rfloor \geq \frac{R-1}{2}$. We want the doubling dimension to be larger than β, which can be easily achieved by choosing R such that $\frac{R-1}{2} > \beta$. Let $R = 2\beta + 2 > 2\beta + 1$. Further, we can now set $\gamma = (2R+5)(k-1)/c = (4\beta+9)(k-1)/c$ and $b = (2R+1)\left\lceil \frac{2R+1}{2} + 1 \right\rceil = (4\beta+5)\left\lceil \frac{4\beta+5}{2} + 1 \right\rceil$. This ensures that the doubling dimension is strictly larger than β.

It remains to be shown that when such a small square contains b nodes, the graph constructed above occurs with probability $p > 0$. The number m of mini-squares contained in a small square of side γr is $m = \frac{\gamma^2 r^2}{r^2/c^2} = \gamma^2 c^2$ which is constant. Each node can fall in any of the m squares with equal probability. Hence, all m^b configurations are equiprobable and $p = \frac{1}{m^b} > 0$. ☐

We number the small squares from 1 to $m = \frac{n}{(\gamma r)^2} = \frac{n}{\gamma^2 \log^{1-\theta} n}$ and denote by X_i^b the indicator variable that takes value 1 when small square i contains exactly b nodes.

Lemma 6.2. *There are at least $n^{1/2}$ squares containing b nodes with probability at least $(1 - O(\frac{1}{e^{n^{0.25}}}))$ for n sufficiently large*

Proof.

$$\begin{array}{ll}\mathcal{E}\left[X\right] & = \mathcal{E}\left[\sum_{i=1}^{m} X_i^b\right] \\ & = \sum_{i=1}^{m} \mathcal{P}\left[X_i^b\right] \\ & = \sum_{i=1}^{m} \binom{n}{b}(\frac{1}{m})^b(1-\frac{1}{m})^{n-b} \\ & \geq m(\frac{n}{b})^b(\frac{1}{m})^b(1-\frac{1}{m})^n \\ & \geq \frac{n}{b^b}(\gamma^2 \log^{1-\theta} n)^{b-1}(1-\frac{1}{m})^{m\gamma^2 \log^{1-\theta} n} \\ & \geq \frac{n}{b^b}(\gamma^2 \log^{1-\theta} n)^{b-1} \frac{1}{2^{2\gamma^2 \log_2^{1-\theta} n / \log_2^{1-\theta} e}} \\ & \geq O(n^{1-O(\frac{1}{\log^{\theta} n})}) \\ & \geq O(n^{\delta})\end{array}$$

where $\delta \geq \frac{7}{8}$ for n sufficiently large, since $\theta > \zeta$.

Let S_i be the random variable representing the small square into which the i^{th} node falls. Let F be the number of small squares containing exactly b nodes after all nodes have been placed. Then the sequence $Z_i = \mathcal{E}\left[F|S_1, ..., S_i\right]$ is a Doob Martingale. One can show that $F = f(S_1, S_2, ..., S_n)$ satisfies the Lipschitz condition with bound 1. Indeed, changing the placement of the i^{th} ball can only modify the value of F by at most 1. We therefore obtain:

$$\mathcal{P}\left[|F - \mathcal{E}\left[F\right]| \geq n^{5/8}\right] \leq 2e^{-2n^{10/8-1}} = 2\frac{1}{e^{2n^{1/4}}}$$

by the Azuma-Hoeffding inequality. Consequently,

$$\begin{array}{ll}\mathcal{P}\left[F < n^{1/2}\right] & < \mathcal{P}\left[F < \underbrace{\mathcal{E}\left[F\right] - n^{5/8}}_{=n^{7/8}-n^{5/8}>n^{1/2}}\right] \\ & \leq 2\frac{1}{e^{2n^{1/4}}} \leq 2\frac{1}{e^{n^{1/4}}}\end{array}$$

and

$$\mathcal{P}\left[F \geq n^{1/2}\right] \geq (1 - 2\frac{1}{e^{n^{1/4}}})$$

□

It now remains to show that in this regime, $\mathcal{G}(n, r)$ are not doubling with high probability.

Theorem 6.2. *$\mathcal{G}(n, (\log n)^{\frac{1}{2}-\frac{\theta}{2}})$, where $\theta \in \,]\,\zeta, 1\,[$ and ζ is a constant such that $0 < \zeta < 1$, are not* doubling *with high probability.*

Proof. By Lemma 6.1, for any constant β, a small square area of side γr with b nodes contains a graph of doubling dimension $> \beta$ with probability $p > 0$. By Lemma 6.2, there are $n^{1/2}$ such small squares containing b nodes w.h.p. Let F denote the number of small squares containing exactly b nodes. Consequently, the probability that at least one of this squares contains a graph of doubling dimension $> \beta$ is given by:

$$\begin{aligned}\mathcal{P}\left[\text{not doubling}\right] &= \textstyle\sum_{j=1}^{m} \mathcal{P}\left[\text{not doubling}|F=j\right]\mathcal{P}\left[F=j\right] \\ &\geq (1-O(\tfrac{1}{e^{n^{0.25}}}))\textstyle\sum_{j=n^{1/2}}^{m}(1-(1-p)^{j}) \\ &\geq (1-(1-p)^{n^{1/2}})(1-O(\tfrac{1}{e^{n^{0.25}}})) \\ &\geq (1-(1-p)^{n^{1/2}})^{2} \\ &\geq (1-\tfrac{2}{x^{O(n)}})\end{aligned}$$

where $x = (\frac{1}{1-p}) > 1$. Consequently, with probability at least $(1-\frac{2}{x^{O(n)}})$, there exists no constant which bounds the doubling dimension of $\mathcal{G}(n, (\log n)^{\frac{1}{2}-\frac{\theta}{2}})$.
□

A-3 Locality-Sensitive Hashing

A family of hash functions $\mathcal{H}$ is said to be locality-sensitive (see [IM98]) if:

Definition 6.1. *We call a family of hash functions $\mathcal{H}$, "(r, cr, p, P)-locality-sensitive" if*

$$\mathcal{P}\left[h(q)=h(u)|d(q,u)<r\right] > p \text{ and } \mathcal{P}\left[h(q)=h(v)|d(q,v)>cr\right] < P$$

Then, we can show the following result (from [IM98]). Let $k = \log_{\frac{p}{P}}(2n)$. Define $g(x) = (h_1(x), ..., h_k(x))$, where $h_i(x) \in \mathcal{H}$. We assume that $d(p,q) \leq r$, for some point p. Consider any point p' such that $d(p', q) > cr$. We have (1),

$$P_1 = \mathcal{P}\left[g(p^*) = g(q)\right] \geq p^k$$

and (2)

$$\begin{aligned}P_2 &= \mathcal{P}\left[g(p')=g(p)|g(p^*)=g(q)\right] \\ &= \frac{\mathcal{P}\left[g(p')=g(p), g(p^*)=g(q)\right]}{\mathcal{P}\left[g(p^*)=g(q)\right]} \\ &\leq \frac{\mathcal{P}\left[g(p')=g(p)\right]}{\mathcal{P}\left[g(p^*)=g(q)\right]} \\ &\leq (\tfrac{P}{p})^k \\ &= \tfrac{1}{2n}\end{aligned}$$

Hence, if (1) is true, then (2) is true with probability at least

$$1 - \sum_{p' \text{s.t. } r_q(p')>R} \frac{1}{2n} \geq \frac{1}{2}$$

. Further, by substituting k in P_1, we have $P_1 \geq n^{-\rho}$. Finally, by choosing $l = n^{\rho}$ functions g_j, we ensure that at least one of them satisfies both properties (1) and (2). In order to search for the nearest neighbor of a point q, we compute $g_1(q), ..., g_l(q)$, and return an arbitrary point that falls in the same bin as q for every function g_j. There are at most n^{ρ} such objects. We can then return the closest point to q among these objects. With constant probability, this point will be one of the cr nearest neighbors of q.

Bibliography

[ABC$^+$05] I. Abraham, Y. Bartal, T-H. Chan, K. Dhamdhere, A. Gupta, J. Kleinberg, O. Neiman, and A. Slivkins, *Metric embeddings with relaxed guarantees*, FOCS, 2005, pp. 83–100.

[ADT07] Salman Avestimehr, Suhas Diggavi, and David Tse, *Wireless network information flow*, 2007, arXiv:0710.3781v2 [cs.IT].

[ADT09] ______, *Wireless network information flow: A deterministic approach*, 2009, arXiv:0906.5394v2 [cs.IT].

[AGGM06] I. Abraham, C. Gavoille, A. V. Goldberg, and D. Malkhi, *Routing in networks with low doubling dimension*, ICDCS, 2006, p. 75.

[AI08] Alexandr Andoni and Piotr Indyk, *Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions*, Communications of the ACM **51** (2008), no. 1, 117–122.

[AX03] D. K. Agrafiotis and H. Xu, *A geodesic framework for analyzing molecular similarities*, J. Chem. Info. Comput. Sci. **43** (2003), 475–484.

[BG05] Ingwer Borg and Patrick J.F. Groenen, *Modern multidimensional scaling - theory and applications*, 2nd ed., Springer Series in Statistics, 2005.

[BGJ05] Jehoshua Bruck, Jie Gao, and Anxiao Jiang, *MAP: Medial axis based geometric routing in sensor networks*, Mobicom, 2005, pp. 88–102.

[BHW08] Maria-Florina Balcan, Steve Hanneke, and Jennifer Wortman, *The true sample complexity of active learning*, COLT, 2008, pp. 45–56.

[BMSU99] Prosenjit Bose, Pat Morin, Ivan Stojmenovic, and Jorge Urrutia, *Routing with guaranteed delivery in ad hoc wireless networks*, DIALM, 1999, pp. 48–55.

[BV05] J.-Y. Le Boudec and M. Vojnovic, *Perfect simulation and stationarity of a class of mobility models*, INFOCOM, 2005, pp. 2743–2754.

[CGJ96] D.A. Cohn, Z. Ghahramani, and M.I. Jordan, *Active learning with statistical models*, JAIR **4** (1996), 129–145.

[CGMZ05] T-H. Hubert Chan, Anupam Gupta, Bruce M. Maggs, and Shuheng Zhou, *On hierarchical routing in doubling metrics*, SODA, 2005, pp. 762–771.

[Cla06] K.L. Clarkson, *Nearest-neighbor searching and metric space dimensions*, Nearest-Neighbor Methods for Learning and Vision: Theory and Practice (G. Shakhnarovich, T. Darrell, and P. Indyk, eds.), MIT Press, 2006, pp. 15–59.

[CT06] Thomas M. Cover and Joy A. Thomas, *Elements of information theory*, 2 ed., Wiley, 2006.

[DCKM04] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, *Vivaldi: A decentralized network coordinate system*, Computer Communication Review, vol. 34, 2004, pp. 15–26.

[Dem94] P. Demartines, *Analyse de données par réseaux de neurones auto-organisés*, 1994.

[Dem04] Eugene Demidenko, *Kolmogorov-smirnov test for image comparison*, Computational Science and Its Applications - ICCSA, 2004, pp. 933–939.

[DF08] S. Dasgupta and Y. Freund, *Random projection trees and low dimensional manifolds*, STOC, 2008, pp. 537–546.

[DH08] S. Dasgupta and D.J. Hsu, *Hierarchical sampling for active learning*, ICML, 2008, pp. 208–215.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni, *Locality-sensitive hashing scheme based on p-stable distributions*, SCG, 2004, pp. 253–262.

[Dim03] K. Agrafiotis Dimitris, *Stochastic proximity embedding*, Journal of Computational Chemistry **24** (2003), no. 10, 1215–1221.

[DPH05] S. M. Das, H. Pucha, and Y. C. Hu, *Performance comparison of scalable location services for geographic ad hoc routing*, INFOCOM, March 2005, pp. 1228–1239.

[FGG$^+$05] Q. Fang, J. Gao, L. J. Guibas, V. de Silva, and L. Zhang, *Glider: gradient landmark-based distributed routing for sensor networks*, INFOCOM, March 2005, pp. 339–350.

[FRZ$^+$05] Rodrigo Fonseca, Sylvia Ratnasamy, Jerry Zhao, Cheng Tien Ee, David Culler, Scott Shenker, and Ion Stoica, *Beacon-vector routing: Scalable point-to-point routing in wireless sensor networks*, NSDI, 2005, pp. 329–342.

[FWV07] Damien François, Vincent Wertz, and Michel Verleysen, *The concentration of fractional distances*, IEEE Transactions on Knowledge and Data Engineering **19** (2007), no. 7, 873–886.

[Gav01] C. Gavoille, *Routing in distributed networks*, ACM SIGACT News (2001), 36.

[GK98] P. Gupta and P. R. Kumar, *Critical power for asymptotic connectivity in wireless networks*, Stochastic Analysis, Control, Optimization and Applications (1998), 547–566.

[GK00] P. Gupta and P.R. Kumar, *The capacity of wireless networks*, IEEE Transactions on Information Theory **46** (2000), no. 2, 388–404.

[GLS08] N. Goyal, Y. Lifshits, and H. Schutze, *Disorder inequality: A combinatorial approach to nearest neighbor search*, WSDM, 2008, pp. 25–32.

[GV06] M. Grossglauser and M. Vetterli, *Locating Mobile Nodes with EASE: Learning Efficient Routes from Encounter Histories Alone*, IEEE/ACM Trans. on Networking **14** (2006), no. 3, 457–469.

[HPAC03] T. Heseltine, N. Pears, J. Austin, and Z. Chen, *Face recognition: A comparison of appearance-based approaches*, DICTA, vol. 1, 2003, pp. 59–68.

[IM98] Piotr Indyk and Rajeev Motwani, *Approximate nearest neighbors: Towards removing the curse of dimensionality*, STOC, 1998, pp. 604–613.

[IM04] Piotr Indyk and Jiri Matousek, *Low-distortion embeddings of finite metric spaces*, CRC Handbook of Discrete and Computational Geometry, 2004, Chapter 8.

[Ind99] Piotr Indyk, *Sublinear time algorithms for metric space problems*, STOC, 1999, pp. 428–434.

[Ind04] P. Indyk, *Nearest neighbors in high-dimensional spaces*, Handbook of Discrete and Computational Geometry (J. E. Goodman and J. O'Rourke, eds.), CRC Press, 2 ed., 2004.

[JM02] Vidit Jain and Amitabha Mukherjee, *The indian face database*, 2002, http://vis-www.cs.umass.edu/∼vidit/IndianFaceDatabase/.

[JMB01a] D. B. Johnson, D. A. Maltz, and J. Broch, *Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks*, 2001, Book.

[JMB01b] David B. Johnson, David A. Maltz, and Josh Broch, *Ad hoc networking*, ch. DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks, pp. 139–172, Addison-Wesley, 2001.

[KK00] B. Karp and H.T. Kung, *Gpsr: Greedy perimeter stateless routing for wireless networks*, MOBICOM, 2000, pp. 243–254.

[KL04] Robert Krauthgamer and James R. Lee, *Navigating nets: simple algorithms for proximity search*, SODA, 2004, pp. 798–807.

[KR02] David R. Karger and Matthias Ruhl, *Finding nearest neighbors in growth-restricted metrics*, STOC, 2002, pp. 741–750.

[KRX06] Goran Konjevod, Andrea W. Richa, and Donglin Xia, *Optimal stretch name independent compact routing in doubling metrics*, PODC, 2006, pp. 198–207.

[KS04] Y. Ke and R. Sukthankar, *Pca-sift: A more distinctive representation for local image descriptors*, CVPR, 2004, pp. 506–513.

[KSW04] J. Kleinberg, A. Slivkins, and T. Wexler, *Triangulation and embedding using small sets of beacons*, FOCS, 2004, pp. 444–453.

[KV02] S. R. Kulkarni and P. Viswanath, *A deterministic approach to throughput scaling in wireless networks*, ISIT, 2002, p. 351.

[KWZZ03] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger, *Geometric ad-hoc routing: Of theory and practice*, PODC, July 2003, pp. 63–72.

[LG94] D. Lewis and W. Gale, *A sequential algorithm for training text classiers*, ACM SIGIR, 1994, pp. 3–12.

[LJDC$^+$00] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris, *Scalable location service for geographic ad hoc routing*, MOBICOM, 2000, p. 120.

[Low04] David G. Lowe, *Distinctive image features from scale-invariant keypoints*, International Journal of Computer Vision **60** (2004), no. 2, 91–110.

[LZ09] Yury Lifshits and Shengyu Zhang, *Combinatorial algorithms for nearest neighbors, near-duplicates and small-world design*, SODA, 2009, pp. 318–326.

[MNP06] Rajeev Motwani, Assaf Naor, and Rina Panigrahy, *Lower bounds on locality sensitive hashing*, SCG, 2006, pp. 154–157.

[MOWW04] Thomas Moscibroda, Regina O'Dell, Mirjam Wattenhofer, and Roger Wattenhofer, *Virtual coordinates for ad hoc and sensor networks*, DIALM-POMC, October 2004, pp. 8–16.

[MR95] Rajeev Motwani and Prabhakar Raghavan, *Randomized algorithms*, Cambridge University Press, 1995.

[MSZ06] Eytan Modiano, Devavrat Shah, and Gil Zussman, *Maximizing throughput in wireless networks via gossip*, ACM SIGMETRIC'06, 2006, p. 26.

[MU05] Michael Mitzenmacher and Eli Upfal, *Probability and computing: Randomized algorithms and probabilistic analysis*, Cambridge University Press, 2005.

[NS03] James Newsome and Dawn Song, *Gem: graph embedding for routing and data-centric storage in sensor networks without geographic information*, SenSys, 2003, pp. 76–88.

[NSDW$^+$07] Mike Nachtegael, Stefan Schulte, Valerie De Witte, Tom Mélange, and Etienne Kerre, *Image similarity, from fuzzy sets to color image applications*, Advances in Visual Information Systems, 2007, pp. 26–37.

[Pan06] Rina Panigrahy, *Entropy based nearest neighbor search in high dimensions*, SODA, 2006, pp. 1186–1195.

[PR97] C. Perkins and E. M. Royer, *Ad-hoc on-demand distance vector routing*, MILCOM '97 panel on Ad Hoc Networks, 1997.

[PR99] Charles E. Perkins and Elizabeth M. Royer, *Ad hoc on-demand distance vector routing*, WMCSA, February 1999, pp. 90–100.

[RRP$^+$03] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, *Geographic routing without location information*, Proc. ACM Mobicom, 2003, pp. 96–108.

[RS98] Martin Raab and Angelika Steger, *Balls into bins: A simple and tight analysis*, Lecture Notes in Computer Science **1518** (1998), 159.

[RTG00] Y. Rubner, C. Tomasi, and L. J. Guibas, *The earth mover's distance as a metric for image retrieval*, International Journal of Computer Vision **40** (2000), no. 2, 99–121.

[Set09] Burr Settles, *Active learning literature survey*, Computer sciences technical report, University of Wisconsin–Madison, 2009.

[SJ99] Simone Santini and Ramesh Jain, *Similarity measures*, IEEE transactions on Pattern Analysis and Machine Intelligence **21** (1999), no. 9, 871–883.

[SMS06] G. Sharma, R. Mazumdar, and N. Shroff, *Delay and capacity trade-offs in mobile ad hoc networks: A global perspective*, INFOCOM, 2006, pp. 1–12.

[Spa02] Libor Spacek, *Face recognition data of the university of essex, uk*, 2002, http://cswww.essex.ac.uk/mv/allfaces/index.html.

[SR06a] G. Schaefer and S. Ruszala, *Image database navigation on a hierarchical mds grid*, Pattern Recognition, 2006, pp. 304–313.

[SR06b] Gerald Schaefer and Simon Ruszala, *Hierarchical image database navigation on a hue sphere*, Advances in Visual Computing, 2006, pp. 814–823.

[SRZF04] Y. Shang, W. Ruml, Y. Zhang, and M. Fromherz, *Localization from connectivity in sensor networks*, IEEE Transactions on Parallel and Distributed Systems **15** (2004), no. 11, 961.

[Tal04] Kunal Talwar, *Bypassing the embedding: algorithms for low dimensional metrics*, STOC (Chicago, IL, USA), 2004, p. 281.

[TC03] L. Tang and M. Crovella, *Virtual landmarks for the internet*, Proc. ACM Sigcomm, 2003, pp. 143–152.

[TDGW07a] D. Tschopp, S. Diggavi, M. Grossglauser, and J. Widmer, *Robust geo-routing on embeddings of dynamic wireless networks*, INFOCOM, 2007, p. 1730.

[TDGW07b] D. Tschopp, S. Diggavi, M. Grossglauser, and J. Widmer, *Robust Routing for Dynamic Wireless Networks Based on Stable Embeddings*, Proc. Information Theory and Applications workshop (ITA) (San Diego, CA), January 2007.

[TP91a] M Turk and A. Pentland, *Eigenfaces for recognition*, Journal of Cognitive Neuroscience **3** (1991), 72–86.

[TP91b] M. Turk and A. Pentland, *Face recognition using eignefaces*, CVPR, 1991, pp. 586–591.

[ÖLT07] Ayfer Özgür, Olivier Lévêque, and David Tse, *Hierarchical cooperation achieves optimal capacity scaling in ad hoc networks*, IEEE Trans. Inf. Theory **53** (2007), 3549–3572.

Dominique Tschopp

Place de la Cathédrale 5
CH-1005 Lausanne
+41 21 311 64 76 / +41 79 772 82 74
dominique.tschopp@gmail.com

Strengths

- EPFL **Diploma in Engineering** and **Ph.D**.
- Expert in **algorithm design and analysis** and **computer networking**
- Native **French** and **Swiss German speaker,** fluent in **English**

Education

- **Ph.D.** in Computer and Communication Sciences / EPFL, Lausanne — Exp. 2009
- **Diploma** (Masters degree) in Communication Systems Engineering / EPFL, Lausanne — 2004
- Academic Exchange Year / KTH, Stockholm — 2001 - 2002

Work Experience

- Research and Teaching Assistant / **EPFL**, Lausanne — 2004 - 2009
  The first part of my thesis work consisted in analyzing a new routing protocol for wireless networks through simulations. The results led to a *European Patent.* In a second phase, I worked on theoretical aspects of algorithms for search and routing. My work was *published in the proceedings of top international conferences.*
  → publication list available at http://people.epfl.ch/dominique.tschopp
- Intern / **IBM Research**, Zurich — 2004
  I performed a 6 months internship for my diploma thesis. I developed and implemented a model for systems-on-chips and proposed a mechanism for failure recovery based on the simulation results. *The work was later presented at an international workshop.* I was also selected to participate in *an IBM EMEA Top student award* event in Nice, France.
- Intern / **International Federation of Red Cross and Red Crescent** (IFRG), Geneva — 2003
  I performed a summer internship at IFRG. I worked on various system administrator tasks. My main contribution was the implementation of a web platform to help visualize and aggregate information about server loads. The results were presented to the IT department of the IFRG, and *raised awareness on unsuspected issues.*
- Intern / **Swiss Institute For Experimental Cancer Research** (ISREC), Lausanne — 2002
  I performed a summer internship at ISREC. I designed and implemented algorithms for DNA sequence alignment

Skills

- Programming
  During my studies and my internships, I worked on numerous programming projects. I used *Matlab* extensively for my research and Java for computer networking projects and teaching. I worked with *php/mysql* for web design. Other programming languages that I worked with include *C, ASP, VHDL, Perl and UML.*
- Problem solving/ Structuring/ Analysis
  During my Ph.D., my theoretical research taught me to understand and solve complex problems and propose sound, efficient and provably good solutions, in a structured way.
- Soft Skills
  Fast Learner / Good presentation skills / Organized / Efficient

Languages

French	Mother Tongue		I grew up in French-speaking Switzerland	
Swiss German	Mother Tongue		I speak Swiss German with both my parents	
English	Fluent	(C2)	I lived in the USA for 6 months, TOEFL 670/677	1998
German	Good	(C1)	Excellent understanding, good writing	
Swedish	Conversational	(A2)	I lived in Sweden for 1 year	2001-2002

Leisure

- Skiing / Tennis / Running / Swimming
- Photography
- Member of the EPFL Tango Association

Status

Date of birth	September 10th 1980
Citizenship	Switzerland
Civil Status	Married