Greedy Routing and Virtual Coordinates for Future Networks

Inauguraldissertation

zur
Erlangung der Würde eines Doktors der Philosophie
vorgelegt der
Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Basel

von

Ghazi Bouabene
aus Tunesien

Basel, 2012

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
auf Antrag von

Prof. Dr. Christian F. Tschudin, Universität Basel, Dissertationsleiter
Prof. Dr. Guy Leduc, Université de Liège, Korreferent

Basel, den 18.09.2012

Prof. Dr. Jörg Schibler, Dekan

Abstract

At the core of the Internet, routers are continuously struggling with ever-growing routing and forwarding tables. Although hardware advances do accommodate such a growth, we anticipate new requirements e.g. in data-oriented networking where each content piece has to be referenced instead of hosts, such that current approaches relying on global information will not be viable anymore, no matter the hardware progress. In this thesis, we investigate greedy routing methods that can achieve similar routing performance as today but use much less resources and which rely on local information only. To this end, we add specially crafted name spaces to the network in which virtual coordinates represent the addressable entities. Our scheme enables participating routers to make forwarding decisions using only neighbourhood information, as the overarching pseudo-geometric name space structure already organizes and incorporates "vicinity" at a global level.

A first challenge to the application of greedy routing on virtual coordinates to future networks is that of *routing dead-ends* that are local minima due to the difficulty of consistent coordinates attribution. In this context, we propose a routing recovery scheme based on a multi-resolution embedding of the network in low-dimensional Euclidean spaces. The recovery is performed by routing greedily on a blurrier view of the network. The different network detail-levels are obtained though the embedding of clustering-levels of the graph. When compared with higher-dimensional embeddings of a given network, our method shows a significant diminution of routing failures for similar header and control-state sizes.

A second challenge to the application of virtual coordinates and greedy routing to future networks is the support of "customer–provider" as well as "peering" relationships between participants, resulting in a differentiated services environment. Although an application of greedy routing within such a setting would combine two very common fields of today's networking literature, such a scenario has, surprisingly, not been studied so far. In this context we propose two approaches to address this scenario.

In a first approach we implement a path-vector protocol similar to that of BGP on top of a greedy embedding of the network. This allows each node to build a spatial map associated with each of its neighbours indicating the accessible regions. Routing is then performed through the use of a decision-tree classifier taking the destination coordinates

as input. When applied on a real-world dataset (the CAIDA 2004 AS graph) we demonstrate an up to 40% compression ratio of the routing control information at the network's core as well as a computationally efficient decision process comparable to methods such as binary trees and tries.

In a second approach, we take inspiration from consensus-finding in social sciences and transform the three-dimensional distance data structure (where the third dimension encodes the service differentiation) into a two-dimensional matrix on which classical embedding tools can be used. This transformation is achieved by agreeing on a set of constraints on the inter-node distances guaranteeing an administratively-correct greedy routing. The computed distances are also enhanced to encode multipath support. We demonstrate a good greedy routing performance as well as an above 90% satisfaction of multipath constraints when relying on the non-embedded obtained distances on synthetic datasets. As various embeddings of the consensus distances do not fully exploit their multipath potential, the use of compression techniques such as transform coding to approximate the obtained distance allows for better routing performances.

Acknowledgements

I would like to thank many people who contributed to this work in various ways.

First, I would like to thank my advisor *Prof. Dr. Christian Tschudin* for the exploration freedom he gave me and for his confidence during the years. Also, I would like to thank *Dr. Christophe Jelger* with whom I had the chance to work during the first years of my Ph.D. Our close collaboration on the ANA EU project was not only fun, but also an occasion to revisit the basics of computer communications, from both a design and a software point of view.

A special thanks goes also to *Dr. Manolis Sifalakis* with whom I had the chance to collaborate during the last years of my Ph.D. His tireless positive thinking helped me concretize many of the ideas presented in this work.

I would also like to thank the following colleagues, for some, close friends, for making this journey memorable and fun: *Paola Ranaldi*, *Diego Milano*, *Marcel Lüthi*, *Michael Springmann*, *Nenad Stojnic*, *David Adametz*, *Ihab Al Kabary*, *Julia Vogt*, *Matthias Amberg* and *Filip Brinkmann*. Some of the lunch time discussions we had are simply unforgettable.

On the private side, I would like to thank *Simone Haselier* for sharing this journey with me, and I am very grateful to my parents and my brothers for their love and caring advice.

Contents

Chapter 1

Introduction

At the core of the Internet, routers are continuously struggling with ever-growing routing and forwarding tables that store control information about every participant network. Such a large control storage is due to the lack of structure of the Internet IPv4 and IPv6 address space, manifested by its fragmentation and the distortion between the hierarchical addressing scheme and the actual graph connectivity.

Until the late 1990's, the number of forwarding table entries for core domain routers was growing exponentially, putting the very functioning of the network at risk as some of the older deployed routers were suspected not to be able to cope with the large amount of data. Since then, with the adaptation of the *Classless Inter-Domain Routing (CIDR)* address aggregation scheme [92] and the progress of routing hardware [52], the threat of growing routing and forwarding tables seems to be under control.

In our view, the resolution of such a problem through a hardware approach can be quite short-sighted with regard to the changes that could affect the network. Indeed, the predictions in [52], comparing the routing table growth to the hardware evolution rate, are mainly based on the connectivity network growth model thus assuming that the network usages will continue to be the same relying on the same architectures and following the same paradigms. When considering however the latest network developments, one can see that such an assumption is deemed to to be wrong. When taking the upcoming data-oriented paradigm for communication [59, 68] as an example, one can already get a feeling of the consequences of such a global-knowledge scheme. In this proposed paradigm of communication, the network participants express their interests in reaching a data element rather than a network host as in the classical host-oriented protocol. Thus the destination "addresses" as well as the routing data allowing to forward incoming requests must be based on a description of the desired data. In such a context, a pure and simple extension of the current global-knowledge based scheme for routing and forwarding to the data-centric case would be simply not feasible. Such an extension would require each multi-homed network participant to maintain an index on the location of every existing piece of information in the global network. Even if one could extend the mechanisms of address aggregation to the names of data, when considering the volumes of nowadays user generated content, the index sizes are likely to go far beyond any current size of host-based indexes. One might also note that the namespace fragmentation effects in the data-centric case would be far more dramatic than in the case of IPv4 or IPv6, as several

distant network nodes can hold similar copies of a piece of data.

In this thesis, we advocate a move away from such global-knowledge based methods and propose to incorporate local-knowledge based algorithms within future network designs in order to avoid such eventual bottlenecks. As a candidate local-knowledge based algorithm, we propose the technique of greedy routing based on virtual coordinates.

1.1 Definition of greedy routing on virtual coordinates

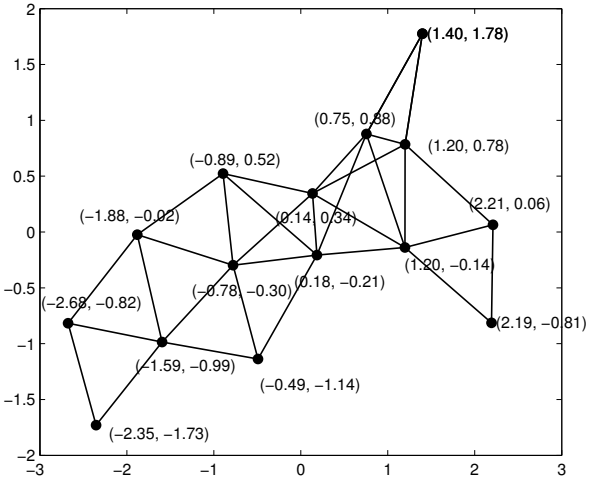


Figure 1.1: Example of virtual coordinates attribution reflecting network proximity

In the general greedy routing method, the main goal is to achieve a successful routing process by having each participant store only a partial knowledge base on the network. As we will discuss in chapter 2, such a lightweight operation cost comes at the condition of a structuring of the address-space. Such a structuring allows to infer routing decisions based (either totally or partially) on operations on the local and destination addresses. In the particular case of greedy routing on virtual *geometric*

coordinates, the "addresses" attributed to the network participants are of a geometric nature, usually corresponding to geometric points in some d-dimensional Euclidean space. Note that the attributed coordinates do not need to correspond to physical coordinates but are rather reflecting positions in the network graph. An example of such coordinates in a two-dimensional Euclidean space is shown in figure 1.1.

Given an incoming packet with a destination coordinate C_d, greedy geometric routing at node u proceeds as follows :

(i) u computes the distance between its own address and the destination address $D_e(C_d, C_u)$ where C_u are the coordinates attributed to node u and D_e is the norm used within the geometric space (usually chosen to be the Euclidean distance)
(ii) u computes the distance $D_e(C_i, C_d)$ for all direct neighbours i of u
(iii) Let j be the neighbour with the smallest distance $D_e(C_j, C_d)$:
    (i) u forwards the packet to j only if
    $$D_e(C_j, C_d) < D_e(C_u, C_d)$$
    (ii) Otherwise, consider that the packet reached its final destination at node u

Hence, the only control information required to perform greedy routing in such a context where addresses encode geometric coordinates, is the list of addresses of the one-hop neighbours in the network graph. This empowers scalability and clearly contrasts with current stateful routing solutions that require to maintain routing information for (sets of) nodes far beyond the immediate neighbourhood.

1.2 The power of geometric virtual coordinates

Our interest in greedy routing on virtual geometric coordinates, is not only due to the scalability offered by the local-knowledge based functioning, but also by the evolutionary power of virtual geometric coordinates and the corresponding vector spaces. Indeed, as will be seen in chapter 2, several approaches to greedy routing do exist. However, it is the particular variant based on geometric coordinates that we are

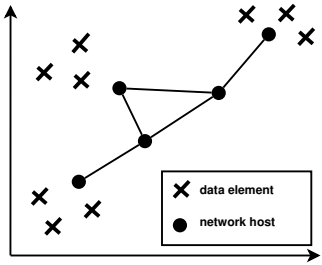


Figure 1.2: Idealized example of a data-oriented greedy embedding : host coordinates reflect the type of data they offer access to (or lie on a path to).

mainly interested in. Our interest is motivated by the high generalization power that lies behind the notion of virtual coordinates as well as the range of potential future applications made possible by the vector space representation.

1.2.1 Virtual coordinates as an evolution tool

Simply put, our vision is to position virtual coordinates to be the common simple abstraction on which future network designs would agree. By designing network functionality to operate on an abstract entity such as a virtual coordinate, we believe that we can achieve a highly evolvable infrastructure by simply modifying the *meaning* behind the virtual coordinates. In order to implement a new functionality into a deployed network, one would simply be required to express the problem in terms of virtual coordinates and "feed" these new coordinates to the deployed system. Such a problem expression in terms of coordinates is the main reason behind our choice of vector spaces as a host for the virtual coordinates. Indeed, given the plethora of mathematical and algorithmic tools that such a space offers, it is the most likely one to accommodate different types of applications.

As an example to demonstrate the flexibility of such an approach, let us imagine an already deployed network that relies on greedy routing on top of virtual coordinates. In this case, the virtual coordinates are attributed to network hosts so as to indicate their proximity at the network level, as it is the case in figure 1.1. When considering the current communication paradigm, such a deployed infrastructure fits into the classical category of host-oriented networking in which the source and destination coordinates indicated in the packet headers correspond to

network hosts.

Imagine now that due to the current usages of the network, we wish to switch the paradigm to a data oriented one in which we are indifferent to the provider of the data. In such a paradigm, users would be indicating interest in data element and thus the source and destination coordinates in packet headers should also reflect such an interest in data instead of hosts. In order to integrate such a (radical) shift to to the deployed infrastructure, it would be sufficient in our approach to express the problem of data centric routing in terms of virtual coordinates. The transition is then made transparent by modifying the meaning behind the virtual coordinates values in order to indicate proximity in terms of data instead of hosts an then feeding the new virtual coordinates to the deployed infrastructure instead of the old ones.

A sketch of such a transition is as follows. First, virtual coordinates could be attributed to the data elements so as to reflect their similarity. Note that the notion of data similarity can be very wide as it can be measured based on the data names, structural properties or content. The data coordinates can therefore be obtained through various methods such as locality sensitive hashing on either the data names or the content [50], or by applying feature extraction methods on the data content [115]. Here, one can already see the advantage of relying on vector spaces for the choice of virtual coordinates as one can preserve the wide spectrum of data proximity representation by simply leveraging several ready-made tools developed within other computing fields.

The rest of the task then is to perform an alignment of the network coordinates to the data ones by attributing coordinates to the hosts so as to reflect their proximity to the data coordinates. Note that the proximity of a host to a data in this case does not only depend on it storing the data but also on the fact that it lies on a path towards a host storing the data. A sketch example of coordinates attribution reflecting such a proximity is given in figure 1.2. In this case, the points indicated by a cross are the virtual coordinates representing data elements while the points symbolized by a dot represent network hosts. As it can be seen from the data coordinates scattering, depending on the coordinates attribution mechanism, data items would tend to be represented as clusters. In order to find the dot coordinates corresponding to the network hosts, one could imagine using a reachability advertisement mechanism in which each node storing a specific piece of data (or cluster in case of aggregation), would announce its presence in a similar way to BGP's path advertisements. Hosts can thereby discover their net-

work distance to particular pieces of data and be assigned coordinates reflecting such a proximity through, for example, a network of springs emulation (where the host coordinates would be pulled towards their offered data' coordinates) or any of the methods described in chapter 2.

A major advantage of a coordinate system such as in figure 1.2, is that it could allow to leverage the power of greedy routing to perform data-centric routing thus avoiding the storage at every participant node of a global data index. This could be achieved by using the same principle for routing on data coordinates: given a destination data coordinate, a node can simply verify which of its neighbours is the closest to the destination by verifying the geometric distance between the coordinates attributed to the neighbour (dot points in figure 1.2) and those of the desired destination data. In a sense, such an approach could be seen as a dual to coordinates based Distributed Hash Table (DHT) proposals such as [90, 91]. Indeed in these proposals, the network hosts are first attributed coordinates so as to reflect their network vicinity. Each data element is later-on hashed to be attributed virtual coordinates and then stored on the network host closest to the obtained coordinates. The data-oriented example sketched above would therefore be the reverse of such a scheme as it is the data elements that are first attributed the coordinates according to their similarity and the network host coordinates are later-on computed as a function of their data proximity.

The biggest advantage however of the computed virtual coordinates attributed to the data points is that they can be easily integrated within the deployed network nodes guaranteeing a smooth network evolution. Indeed given that the network participants already implement the greedy routing functionality, it is sufficient to replace the older control state consisting of the virtual coordinates based on network locality, by the newly obtained coordinates representing data proximity. As a result, we could in theory modify the behaviour of the network with a complete shift in the networking paradigm without changing the core deployed functionality. Each participant node (router) can simply continue to operate in the same greedy way: given a destination coordinate, a list of neighbours coordinates and its own coordinates, a node simply forwards the packet to the node closest to destination.

Naturally, such a scenario is an idealized case, and such a network paradigm switch would in reality incur more difficulties. It is our belief however that although the usage of virtual geometric coordinates would not completely resolve all the transition challenges, it would certainly alleviate a large number of them. When contrasting such a smooth

transition with the current efforts for the deployment of a data centric network [88] where the driving idea is the development of a fully new protocol suite one can see that our approach would ease the deployment of new functionality to the production network as it would more easily comply with already deployed infrastructure.

Moreover, and as an additional argument for the flexibility of greedy routing on virtual coordinates, an example scheme such as above could be further extended to support the co-existence of both networking paradigms within the same deployed infrastructure. In fact, given that the required network functionality can be factorized, one could simply extend the deployed functionality to support the presence of different types of virtual coordinates. Each node would then have to maintain several parallel "realms" of virtual coordinates, each corresponding to a different functionality as in the case of host-oriented and data-oriented routing. Not only would such a scheme allow for the co-existence and thus competition (in a positive sense) of different networking styles, it would also allow an easy collaboration between the different paradigms. To continue on the example corresponding to figure 1.2, one could for example imagine using a communication relying on both paradigms : greedily routing the request based on the data-oriented coordinates, while transmitting the response (the actual data transfer) via greedy routing on network-proximity-based coordinates, thus allowing for a faster data transit.

In contrast, most of the future network architecture proposals that target to achieve the same diversity in the networking functionality [79] rely on the concept of *virtualization*. In this case, the main idea is to share the resources of the deployed routers and switches among different network architectures (styles, paradigms) by running several virtual machines, each implementing the desired functionality. Although such a scheme offers the infrastructure for the easy deployment of novel networking paradigms as well as for their "peaceful" coexistence, it does not favour the *collaboration* between the different "virtual" networks. By having each proposal boxed into a virtual machine, the probability of an interaction at a higher level (i.e. above the virtual machine) similar to the one proposed above is rather low. In our case however, due to the interconnection between the different realms (compartments) at the functional level (by for example sharing the greedy routing functionality), such a collaboration would be made much easier.

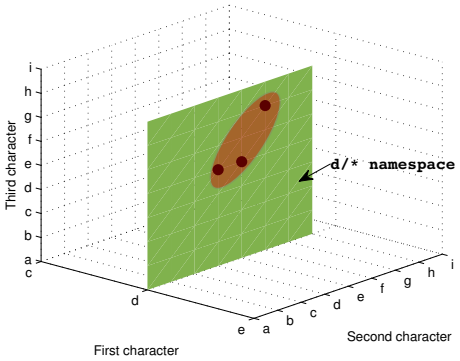


Figure 1.3: Visualization of hierarchical name aggregation : announced namespace is too wide. When represented in a vector space, compacter aggregations such as the red ellipsoid would be possible

1.2.2 Applications of virtual coordinates and vector spaces

In addition to the evolution capability brought by their abstract nature, virtual coordinates, when restricted to vector spaces, also offer the advantage of their structure and the variety of algorithmic and mathematical tools that can apply on them. Applying such tools on networking problem can allow for easier and better solutions.

As an example, one of the typically recurrent needs in communication protocols is the possibility of aggregation of the address elements. Such a feature is not only used while storing control information to reduce the information load, but also as a part of "reachability announcement" protocols in order to reduce the amount of control messages. An example of such a reachability protocol are path-vector protocols announcing the reachability of hosts, data or services. A dominant approach for achieving aggregation in the networking literature are hierarchical name-spaces of which IP addresses, File System names and URLs are typical examples. A major disadvantage of using such systems however is their coarse-grained nature.

Taking again a data reachability announcement example (in this case it might as well be on top of a host-oriented paradigm), imagine a host that wishes to announce itself as a provider for three different files, named : "d/de", "d/ee" and "d/fg". Note that we restricted the file

names in this case to be three-character sized (not counting the slash sign), so as to be able to represent the files in a three-dimensional space as in figure 1.3. When announcing the files' presence to its neighbouring nodes, the node can naturally specify all their three names. If it wishes however to aggregate the announcements according to the hierarchical naming, the natural choice would be to announce the presence of files "d/*" where the wildcard "*" means any number of any valued characters. When visualized in a three-dimensional space where each dimension indicates the variation of one of the characters, such an announcement would be equivalent to the plane in figure 1.3, orthogonal to the first axis at value "d" in which the first letter is fixed to value "d" while the other two values are free. Such an announcement would be an overkill in our view as it would announce a far too wide address-space thus inducing other network participants into requesting non-serviced files. When considering however the possibility of using geometric-space-specific tools to perform the aggregation, one can easily find a more fine-grained representation for the three different points as is the case of the red ellipsoid in figure 1.3. In fact when abstracting the file names as points in a multi-dimensional Euclidean space, several tools from the *Machine Learning* and more particularly *clustering* as well as from the *database information retrieval* literatures can be leveraged to perform such an aggregation.

Another advantage of such an easy address-space aggregation is the possibility of fine-grained *geocasting* [66] that is the equivalent of a broadcast limited to a particular region of the address-space. Hence, similar to the announcement case, leveraging analytical tools to define more fine-grained namespace regions could allow for a more efficient geocasting.

Another nice application emerging from the use of virtual geometric coordinates is that of trajectory-based routing [81, 82] that allows the sender node to indicate a desired path to be followed by its packets. In classical datagram oriented communication, the dominant approach so far to do so is that of *source-routing*. In this approach, the sender simply indicates the sequence of hops to be taken by its transiting packet. Such a mechanism requires however a prior network knowledge from the sender and incurs a load on the packet header as it requires storing several node identifiers. When taking advantage of the analytical nature of a vector-based address space, one could simply formulate a function approximating the path to be followed by the transiting packet. Such a scheme would neither require a prior knowledge of the network node

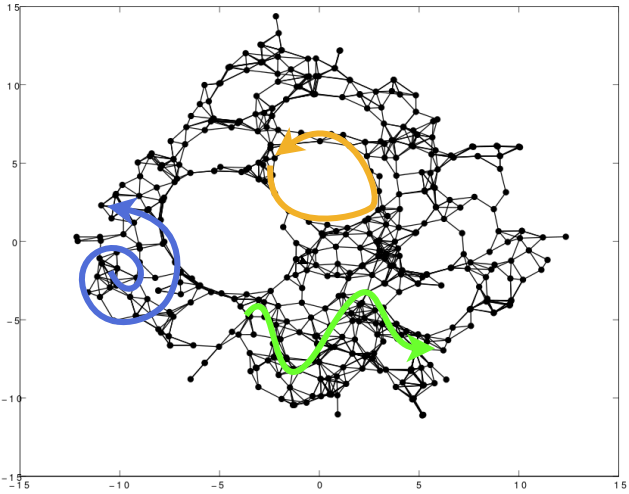


Figure 1.4: Example usages of trajectory based routing : blue spiral for service announcement, orange loop (boomerang) for monitoring, Green curve to avoid given transit points

identifiers nor incur a load on the packet header as compact path formulations would be possible such as polynomial functions [82] or parametric curves [81]. As pictured in figure 1.4, such a capability could for example be used for service announcements, by sending a message on a spiral trajectory, for network monitoring by sending a control message on a *boomerang* trajectory, or simply for incurring path deviations towards a destination, due to security or legislation reasons for example.

1.3 Challenges to the deployment of greedy routing on virtual coordinates in future networks

In our view, the greedy routing on virtual coordinates technique can be considered as promising candidate component for future network, due to all the possible usages described in the sections above. However, in order

to allow this technique to convince future networks design communities to make the jump towards it, the minimum pre-requisite is to make it capable of correctly achieving *today's* networking requirements.

A first major challenge to the application of greedy routing on virtual coordinates is that of *routing dead-ends*. These are local-minima to the distributed greedy routing algorithm that are due to the difficulty of virtual coordinates assignment. In fact, in the greedy routing algorithm depicted on page 4, two notable aspects are worthy of further attention. First, the *distance minimising* condition at step *3.a* is key to the stability of geometric greedy routing so that packets deviating from the correct path do not end up bouncing from one edge of the network to another in infinite routing loops. Second, reaching the termination condition at step *3.b* does not guarantee that a packet reached its *correct* recipient. Indeed, in some cases an intermediate node might be closer to the destination than any of its direct neighbours, halting in this way the forwarding process. Such a node is a *local minimum* in the greedy routing process. The problem of local-minima has confronted research on geometric greedy routing since the early days, and has been thereafter more or less addressed in proposals that either extend the basic greedy routing heuristic (e.g. perimeter flooding [89] or face routing [63]), or devise sophisticated distance functions and coordinate spaces (e.g. [64], [104], [40]). In this context, we expand the solution space by proposing a routing recovery scheme based on a multi-resolution embedding of the network. By recovery, we mean a mechanism allowing to reset the packet on a correct path in case a greedy routing dead-end is reached. The recovery in our case is performed by routing greedily on a blurrier view of the network. Such a coarser-grained view of the network is obtained by embedding several clustering-levels of the graph in different low-dimensional Euclidean spaces. When compared with higher-dimensional embeddings of the network, our method shows a significant diminution of routing failures for a similar header and control-state sizes.

A second big challenge to the application of virtual coordinates and greedy routing to future networks is the support of administrative relationships between the network participants inducing a differentiated services environment. A typical example of such relationships are the *customer-provider* and *peer-peer* [17] relationships between the Autonomous Systems (AS) providers in the internet that result in some paths being accessible to some customer AS and not to others.

In contrast, geometric greedy routing mechanisms have been mainly

designed for operation in policy-free single-organisation networks or collaborative (under a common policy) communities. In such a case, all the nodes forming the network belong to the same administrative entity and their communication is oblivious to special policy rules such as those expressing administrative (and possibly competing) incentives. For an Internet-scale shared network infrastructure however, it is unlikely that the different parties contributing their resources, will abide to a policy-free access model, if nothing else because of security concerns, traffic engineering and pricing issues. The challenge therefore is to make a simple method such as greedy routing compatible with such complex policies without depriving it from its simplicity property. Although an application of greedy routing within such a setting would combine two very common fields of today's networking literature, such a scenario has surprisingly not been studied so far. In this context we propose two approaches to solve the problem.

In a first approach, explored in chapter 5, we implement a path-vector protocol similar to that of BGP on top of a greedy embedding of the network. By propagating reachability announcements, each node can then build a spatial map associated with each of its neighbours indicating the address-space regions to which the neighbour offers access to. By viewing the routing decision through a neighbour as an attribution of a destination coordinate to one of the two sets *offered* or *non-offered*, such a decision problem becomes then similar to a classification problem. Routing is then performed in our case through the use of a decision-tree classifier taking the destination coordinates as input. When applied on a real-world dataset, the CAIDA 2004 AS graph [35], we demonstrate a considerable compression of the routing control information as well as a computationally efficient decision process comparable to methods such as binary trees and tries.

By extending the number of classes to not only carry a reachability information but also a distance one, we obtain an equivalent to distance based administrative greedy routing. In fact, although such an approach is built on simple off-the-shelf tools, it is in fact comparable to a metric-learning solution in which each node learns a different distance function, per-neighbour, allowing to better fit to the paths advertised by the neighbour. Such a distance-per-neighbour model is in fact an important step forward in the greedy embedding literature that has been so far struggling to find the destination metric space (Euclidean, Hyperbolic, etc.) to fit at best any network graph.

In a second solution attempt to the problem of administrative greedy

routing, we take inspiration from the social sciences approach by searching for a consensus structure on the network distances. The goal is to find agreed upon values for the source-destination network distances so that they satisfy the different perceptions of the network participants. This is achieved by defining a set of constraints on the inter-node distances so that a successful greedy routing following the administrative rules is guaranteed. When searching for such distances, we take advantage of special properties of the administrative network at hand to encode multipath support in the obtained distances. We demonstrate a good greedy routing performance as well as a good multipath support (and hence fault tolerance) of the obtained distances on synthetic datasets. As various embeddings of the consensus distances do not fully exploit their multipath potential, we also experimentally investigate the possible use of compression techniques such as transform coding to approximate the obtained distance values.

1.4 Summary of contributions

The main contributions of this thesis are the following:

(i) As a design contribution, we propose to promote virtual coordinates within future networks as a *consensus* generic representation of communication network elements (hosts, data, etc.).

(ii) In the context of greedy geometric routing, we propose a novel failure recovery scheme based on a multi-resolution embedding of networks in low-dimensional Euclidean spaces.

(iii) We identify a major challenge to greedy routing algorithms in general (be they geometric or not), that consists in the administrative relationships among participants. We also lay the first main guidelines to address this challenge.

(iv) We demonstrate the advantage of virtual coordinates through the use of classification tools to efficiently address the administrative relationships problem

(v) We propose a method to extend the capability of greedy routing algorithm through a pre-processing of the network distances on which the routing is performed. The simple greedy routing algorithm can then be influenced to support features such as administrative compliance as well as multi-path routing and fault tolerance.

Chapter 2

Obtaining Coordinates for Greedy Routing

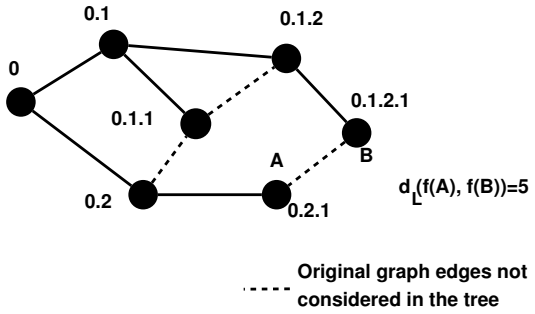


Figure 2.1: Example of node labeling along a spanning tree of the graph allowing for label-based greedy routing

As virtual coordinates attribution to the network nodes play a central role in our work, the goal of this chapter is to introduce this topic to the reader. Note that the range of methods described in this chapter might be quite broad as we attempt to provide a survey of the network labelling techniques. Nevertheless, many of the mentioned methods in this chapter will be referred to and used in the coming chapters. In the following, we reference the possible theoretical methods for virtual coordinates attribution and their limitations as well as the approaches to the problem in the networking literature.

2.1 Theoretical approaches to the problem of virtual coordinates attribution

2.1.1 Distance labelling techniques

Let us start by first defining the problem of virtual coordinates attribution and our expectations from any candidate solution. Given our intention to use the virtual coordinates for greedy routing, an essential task therefore is to be able to determine which of the neighbour nodes is in fact closest to the destination based only on the destination node's coordinates that is included in the packet header, and on the neighbours' coordinates. Note that as implied by the greedy routing algorithm in section 1.1, it is the order of the neighbours distances to the destination that matters more than the actual distance values. Nevertheless, one could assume that when provided with coordinates offering a good

approximation of the original network distances, the order sequence of the distances should in principle also be well represented. Therefore, a common way of approaching the problem of greedy routing on virtual coordinates is to search for coordinates that approximate at best the original network distance values (or an isotonic transformation of the distances preserving their order). Such a problem can be defined as follows. Given an undirected, possibly weighted, graph G, let d_G be the shortest path distance between any two nodes s and $t \in G$. Given a label space L, a node labelling is a function $f : G \to L$ attributing to each node $u \in G$ a label $f(u) \in L$. Let d_L be a distance function defined on the label space L allowing to compute a dissimilarity value between any two given labels $l_1, l_2 \in L$. Our goal of obtaining a good labelling of the graph is then to find a label space L, a mapping function f and a distance function d_L such that :

$$d_G(s,t) \simeq d_L(f(s), f(t)),\ \forall s,t \in G$$

Notice in the definition above the intentional use of the term *label* instead of *coordinates* in order to define the attributes assigned to the graph vertices. This is due to the fact that the coordinates denomination inherently carries a geometric meaning, thus suggesting that the label space L should necessarily be a geometric one. However, and as can be seen in the definition above, there is in reality no such requirement from the greedy routing objective point of view. As far as greedy routing is concerned, the labels can be of any nature: binary, ascii, integer values, a multidimensional vector of floating point values, etc. Also, the distance d_L can respectively be the *Hamming* distance, the *edit (Levenshtein) distance*, or any of the *Minkowski* L_p metrics, as long as the condition of a good distance approximation for all nodes of the graph holds.

A common method in the networking literature for achieving such a non-geometric labelling of the graph vertices allowing an approximation of the graph distances is the use of tree graphs and the tree-distance metric. A typical procedure of doing so is to consider a labelling of one of the spanning trees of the graph with incrementally growing labels where the label assigned to a child node is prefixed by that of its parent as shown in figure 2.1. The distance function between two nodes s and t then used is :

$$d_L(f(s), f(t)) = d_e(f(s), C) + d_e(f(t), C)$$

where d_e is the *edit* or *Levenshtein* distance and C is the longest common prefix label present in both $f(s)$ and $f(t)$ (C is also the at-

tributed label to the closest common ancestor of s and t). Such a simple scheme has however the disadvantage of generating very large sized labels in case the depth of the spanning tree is not well balanced. Another critical disadvantage in such a case is the poor approximation of the original graph distances due to the fact of ignoring some of the edges of the graph. Indeed, given that the graph edges not belonging to the tree cannot be considered when evaluating the distances on the labels, the returned distance is thus the length of the path traversing from the source node towards the closest common ancestor and then down to the destination. Hence in the case of two (originally) neighbour nodes such as nodes A and B in figure 2.1, the tree distance could be a poor approximation of the original graph distance $d_G(A, B)$. The literature of tree-metric embedding [33, 21, 3] targets to alleviate the poor distance approximation problem, by finding a better tree structure (not necessarily a spanning tree of the graph), such that the distances on the chosen tree approximate at best the original graph distances.

More generally, the tree-based methods lie under the umbrella of *Distance Labelling Methods* that in turn rely on general sparse graph spanners (spanners are subgraphs of the original graph including all of the original graph's vertices with fewer edges, such as trees). Being a quite theoretical research topic, the literature in the distance labelling field is mainly concerned with uncovering upper and lower-bounds of both *Exact* and *Approximate* distance labelling.

The *Exact Distance Labelling* literature [49, 86], is a branch of the graph labelling methods that aims at attributing positive integer labels to the nodes of the graph (network) while maintaining the choice of the distance function d_L open, so that for any pair of nodes s and t belonging to the graph, we have the *exact* equality $d_G(s,t) = d_L(f(s), f(t))$. The major challenge for this family of methods is then the size of the attributed labels as well as the complexity of the distance function. To our knowledge, the latest reported lower bound on the size of labels for general graphs is $\Theta(n)$ bits [49].

In the less constrained branch of approximate distance labelling [4, 48, 110], the exact equality condition is relaxed and a *stretch* of the original distances is allowed. Different types of stretch are considered in this literature, such as the *additive* stretch :

$$d_G(s,t) \leq d_L(f(s), f(t)) \leq d_G(s,t) + \beta$$

, the *multiplicative* stretch :

$$d_G(s,t) \leq d_L(f(s), f(t)) \leq \alpha * d_G(s,t)$$

and the *affine* stretch :

$$d_G(s,t) \leq d_L(f(s), f(t)) \leq \alpha * d_G(s,t) + \beta$$

The best approximate distance labelling scheme reported to our knowledge allows for a multiplicative stretch where $\alpha = 2k - 1$, given an integer k, with labels of size $O(n^{1/k} \log n \log (n\Delta))$ bits [110], where n is the number of nodes in the graph and Δ is the diameter of the graph, i.e. the largest distance between any two nodes. Note that the distance estimation cost of this solution is constant at $O(k)$. The labels attributed to the nodes in this case are a combination of unique node identifiers and distance values selected among the full data set so as to guarantee the stretch condition. To get an idea about this result, for an example network such as the Internet Autonomous System Graph, with around 33000 vertices, and a diameter of say 9 hops, in order to have a 3-multiplicative stretch approximation of the distances ($k = 2$), we must use labels of size $O(49572)$ bits. Although such a label size is considerably inferior to the $O(n^2)$ size of a regular distance matrix, it is still inconvenient for applications such as datagram-based routing in which the destination labels are carried within the packet. Therefore, a natural prolongation of the field of approximate distance labelling is that of *Compact Routing* [109, 70] that combines labelling techniques with locally stored routing information called *Distance Oracles* to perform efficient routing while requiring small sized routing tables and labels. The role of the distance oracle data structure in this case is to approximate the distance between any pair of nodes by storing only a subset of the distance values and relying on triangular inequality compliance to induce upper and lower bounds on the distance values.

Another effort to label graph nodes in the 1970's was mainly focused on embedding the graph into a *Hypercube* data structure [53, 10]. In this case, the vertices of the graph were attributed binary labels (corresponding to vertices of a multidimensional hypercube) and the chosen label distance function d_L was the *Hamming distance*. Due to the fact that not all graphs are addressable according to a purely binary alphabet, the authors in [53] adopted a ternary alphabet and a modified Hamming distance that ignores the "bit" differences in case the third (new) symbol is encountered. However, also in this proposal, the major problem remains the size of the attributed labels required for the embedding that is of $O(n-1)$ bits.

As can be seen from the above presented results, with the exception

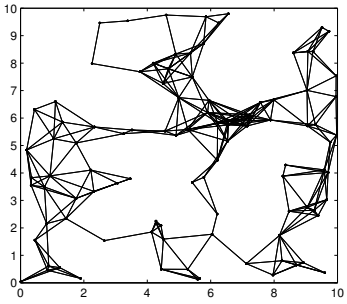
of spanning tree labelling, all the solutions for both approximate and exact distance labelling would require node labels that are too large to be shipped into a packet header. For the exceptional case of the spanning tree labelling methods, the price to pay however is the poor approximation, or stretch, of the original graph distances.

2.1.2 Geometric Approaches

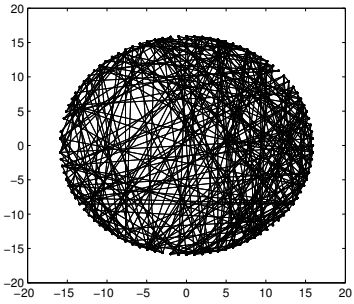
Other graph labelling techniques considered the possibility of embedding the graph into a geometric space meaning that the labels attributed to the vertices of the graph are multidimensional points belonging to a *Hilbert* space for example. Although this might sound limiting when compared with the wider distance labelling goals above, it does however offer several advantages. First, by priorly selecting the target embedding space's characteristics, such as the distance function and the dimensionality, one already fixes the resulting label size and the complexity of the distance approximation procedure. The variable in this case would then be the quality of the obtained embedding. Also, some of the applications from which the original graph was extracted (such as molecular structures or communication networks), might carry a geometric component by nature. This is typically the case of sensor networks where one might be interested in the recovery of sensory data collected within a specific geographic region. Hence, adopting a geometric labelling in such cases would be natural. Generally, graph data analysts recur to geometric embedding methods due to the wide variety of mathematical and algorithmic tools applicable in vector spaces, thus easing the relational data analysis.

Graph drawing techniques

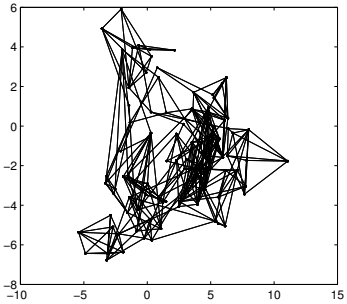
A straightforward way of obtaining geometric coordinates associated with graph vertices is to recourse to graph drawing techniques. These techniques, usually employed to visualize the graph data-structure, associate each graph vertex to a point in a low-dimensional (2 or 3-D) geometric space (usually Euclidean). A line segment between two points is then drawn in the geometric space if the corresponding vertices are related by an edge in the graph. A simple method for obtaining geometric coordinates for the vertices of a graph would then be to *draw it* using one of the techniques presented below, and attribute to each graph vertex the low-dimensional coordinates of its associated point in the drawing as a label.



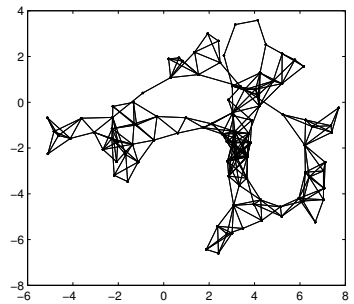
(a) Original generated network using unit-disk graph method with 100 nodes



(b) All geometric points are initialized around a circle at the beginning of the algorithm



(c) Graph drawing after 100 iterations



(d) Graph drawing after 500 iterations (stable state)

Figure 2.2: Example of the Kamada-Kawai springs-network graph drawing

Due to their visual purpose, graph drawing techniques focus more on the aesthetic aspects of the drawing such as planarity and cluttering avoidance, than on the precise representation of the graph distances and might not always provide a good approximation of the original distances.

The most commonly used techniques for graph drawing are the force-based techniques [62, 44, 112]. Force-based graph drawing techniques may vary according to the nature of the physical force they simulate in order to embed the graph. Physical phenomena such as elasticity (of rubber-bands or springs), gravity, electricity and magnetic fields can be simulated in a system corresponding to the graph. The common idea for these drawing techniques, independently of the simulated phenomenon, is that the system corresponding to the graph in the geometric space, namely the set of geometric points and segments representing the vertices and the edges of the graph, would be initiated in a random, highly energetic and highly unstable state. The graph drawing techniques then let the system "evolve" in time, by simulating all the different physical interactions among the geometric points and segments until an equilibrium state is reached, in which the simulated physical system is in a minimal energetic state.

The most widely known technique of force-based graph drawing is that of spring-networks proposed by *Kamada and Kawai* [62]. In this particular model, both elastic and electrical forces are combined in the simulated system. Each geometric point associated to a graph vertex is assumed to carry an electrical charge. Two points associated with graph vertices related by an edge, are also assumed to be *linked* by a mechanical string that has stability properties (*Hook's law*) related to the corresponding graph edge's weight. Typically this means that the length of the spring (edge between two geometric points) at a rest state is proportional to the weight of the edge between the graph vertices associated with those points. An obvious intention of using such a system is to maintain the proximity information between the nodes, by constantly having a force pulling neighbour nodes close to each other. The electrical charges attributed to the geometric points act instead as a repulsive force, pushing two vertices away from each other in case they are too close. The intention of using such forces in this case is to avoid cluttering of the drawing and obtain a better repartition of the points in the space. Such a wide repartition would enhance the readability of the embedding and offer an aesthetically more pleasant drawing.

When started at an initial state by attributing either chosen or random coordinates to the graph vertices, the system will at first behave

quite chaotically, with the spring forces pulling nodes together and the electrical ones pushing them apart. However, with time, (or in a simulation environment through iterations) the system's behaviour will stabilize until finally reaching an equilibrium state in which the total forces applied in the system are minimal and thus the positions of the geometric points are fixed (or there are minimal, insignificant, variations). Figure 2.2 shows an example of the evolution of such a system between the start, middle of the iterations, and the final obtained state. Note that several variations and simplifications of this model exist. Notably, a simplification considering only the spring forces between the points, and not relying on any electrical repulsion forces, results in a system that minimizes the difference between the distance of the points in the geometric space and the distance of their corresponding vertices in the graph. Such a system would then run a simulation having a similar result as a *Multidimensional Scaling* approach described further below.

Such graph drawing methods relying on physical phenomena simulations are very intuitive and thus quite attractive to practitioners. However, a general criticism is that in a centralized execution environment, in which the totality of the graph information is available, a computationally more efficient approach would be to simply formulate the total energy of the simulated system as a function of the points' positions and to minimize it using regular optimization techniques such as gradient descent or non-linear programming methods in general. This would avoid the need of computationally simulating a physical system that would only output a local minimum of the energy function as a result. However, and as will be shown in the applications review section, such a physical system simulation gains in importance when the embedding application is to be distributed among several hosts (computing agents) holding only parts of the total connectivity information (graph).

Spectral decomposition methods

Other graph drawing techniques rely on more analytical approaches to attribute geometric coordinates to the graph vertices. A notable such method, developed within the framework of spectral graph theory is that of the *Laplacian Eigendecomposition* or *Eigenmaps* [7]. This method is also usually categorized under the set of *local* methods for embedding [31] as it is mainly concerned with preserving the proximity of adjacent nodes in the embedding and gives no importance to the distances between non-adjacent nodes.

More Formally, provided that we wish to attribute to each vertex i belonging to the graph G, a d-dimensional vector $\boldsymbol{y_i}= (y_{i1}, y_{i2}, .., y_{id})$ in a Euclidean space, the Laplacian eigenmaps method targets to minimize the following error function :

$$\sum_{i,j} \|\boldsymbol{y_i} - \boldsymbol{y_j}\|^2 W_{ij}$$

where $\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^2$ is the squared Euclidean distance between points $\boldsymbol{y_i}$ and $\boldsymbol{y_j}$ and W_{ij} is a weight value associated with the pair of vertices i and j. In the classical Laplacian eigenmaps approach, also denoted as the "simple-minded" one by the authors [7], the weights are attributed so that :

$$W_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are connected} \\ 0 & \text{otherwise} \end{cases}$$

In this case, the matrix W is then equivalent to the incidence matrix of the graph. Notice in this case the zero-valued weights for non-connected vertices. This means that the inter-point distance for non-connected vertices does not contribute at all to the embedding error function and that instead, this method focuses on maintaining the locality of neighbour nodes only, by making sure that any large distance between two neighbours nodes is penalized (hence the classification of this method among the local methods).

When further expanded and simplified, the error function above is equal to

$$\sum_{i,j} \|\boldsymbol{y_i} - \boldsymbol{y_j}\|^2 W_{ij} = tr(Y^T L Y)$$

where Y is the matrix listing all nodes' coordinates, $tr(X)$ is the trace of a matrix X, namely the sum of its diagonal values and L is known as the *Laplacian matrix* of the graph and is defined as

$$L = P - W$$

where P is a diagonal matrix indicating the nodes' degrees, such that $P_{ii} = \sum_{j \neq i} W_{ij}$. Hence, minimizing the above embedding error function becomes equivalent to

$$\underset{Y^T P Y = 1}{\operatorname{argmin}} \; tr(Y^T L Y)$$

where the constraint $Y^T PY = 1$ is there to ensure that the embedding remains within a d-dimensional space and does not collapse into a lower dimensional space. Such a quadratic form is then minimized by the eigenvectors of the Laplacian matrix L (i.e. the $\boldsymbol{y_i}$ vectors such that $Ly_i = \lambda y_i$ where λ is the eigenvalue associated with the vector). These eigenvectors can be found by performing a Singular Value Decomposition (SVD) on the matrix L. A d-dimensional solution can be found by taking the d eigenvectors of L with the smallest eigenvalues (except for the first one that has eigenvalue 0). Note that the Laplacian matrix is also used for graph-based clustering algorithms such as *spectral clustering* [98] as the eigen-problem resolution above also accounts as a relaxation of the minimum-cut problem in a graph.

Another similar technique relying on eigen-decomposition operations (or Singular Value Decomposition) of matrices related to the graph is that of *Classical Multidimensional Scaling* [12]. As its name indicates, this method is usually classified among the Multi-Dimensional Scaling (MDS) methods detailed further below. However, in our view, such an approach for graph (or more precisely *metric*) embedding bares more similarity with the spectral methods presented here then with the rest of the MDS methods. The Classical MDS method takes as input the shortest path distance matrix D of the graph. Assuming that the targeted embedding is a Euclidean one, in which the matrix Y holds the attributed coordinates of the nodes, then a direct relation between the graph (desired) distance values and the inner-product matrix $K = YY^T$ can be established [12], where the squared distance between two points i and j can be expressed as:

$$D_{ij}^2 = K_{ii} + Kjj - 2K_{ij}$$

thus allowing to extract the matrix K through a double-centring operation

$$K = -\frac{1}{2} J D^{(2)} J$$

where J is a constant matrix so that $J = I - n^{-1}\mathbf{1}\mathbf{1}'$ (and $\mathbf{1}$ is the vector with all components set to 1). By performing an SVD decomposition of the K matrix, one can find matrices Λ and Q such that

$$K = Q\Lambda Q^T = (Q\Lambda^{1/2})(\Lambda^{1/2}Q^T) = YY^T$$

Hence, by taking embedding coordinates matrix $Y = Q\Lambda^{1/2}$, one can find a Euclidean embedding that matches *exactly* the desired graph

distances. However, such a spectral decomposition of the K matrix may lead to several eigenvalues and eigenvectors, thus leading to a very high dimensional Euclidean embedding of the graph. A simple way of obtaining a lower, d-dimensional embedding is to select Y such that it is composed of d eigenvectors of K with the largest eigenvalues. Such an operation is in fact equivalent to a linear dimensionality reduction on the high-dimensional data matrix Y (exact embedding), through Principal Component Analysis. Note that the above performed linear operations on the graph distance matrix allows to find a matrix K minimizing an error function known as *strain*

$$S(Y) = \|XX^T - YY^T\|^2$$

An important remark on the classical MDS approach is that it assumes a direct relation between the node's distances and the inner-product matrix K, as if the distances were necessarily generated in a high dimensional Euclidean space. This might be true for some input distance data, but not quite, when the distance data is generated from other structures such as a graph.

Therefore, the recently proposed method of *Structure Preserving Embedding* (SPE) [102], sets to search for an inner-product matrix (also denoted as *kernel*) K such that it satisfies particular constraints on the embedded distances. More precisely, the goal in the SPE approach is to be able, based solely on the embedded positions and a graph generation algorithm, to reconstruct the original graph. By graph generation algorithm, we mean for example a k-Nearest Neighbours (k-NN) algorithm that, given the position of a point, creates an edge relation to the k-nearest points in the vicinity of the chosen one. Another example of a graph generating algorithm is the ϵ-ball one, adding an edge (connection) between a given point and all points lying within a distance inferior to ϵ from it. When choosing a particular graph generation algorithm such as the ϵ-ball one, in order to be able to reconstruct the initial graph, one must ensure that the distance between non-connected nodes is superior to ϵ. This then defines a set of linear constraints on the embedding distances such that :

$$D_{ij}(A_{ij} - \frac{1}{2}) \leq \epsilon(A_{ij} - \frac{1}{2})$$

where A_{ij} is the incidence matrix of the graph. Due to the direct relation between the distance matrix and the kernel matrix, the set of constraints defined above also apply to the kernel (inner-product) matrix K. Hence

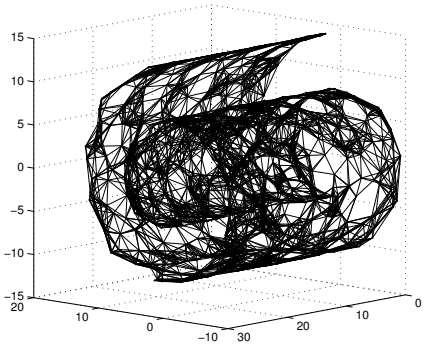


Figure 2.3: Example dataset (known as the Swiss-roll) where the data lies on a lower-dimensional manifold. Generating a k-NN graph on the data, results in an approximation of the manifold

a *better* K matrix in this case is searched so as to satisfy the conditions allowing to correctly reconstruct the graph. The solution for K can be found using a Linear Programming technique known as *Semi-Definite Programming*. Then, an eigendecomposition of the matrix K, similar to the ones performed above, is performed in order to obtain the point coordinates.

Relation to manifold-learning and dimensionality reduction : *Manifold learning* is a sub-field of non-linear dimensionality reduction methods counting methods such as *ISOMAP* [108], *Locally Linear Embedding* [96], and *Maximum Variance Unfolding* [116]. In this case, the input data is a (very) high-dimensional *vector* data. In principle such an input dataset has nothing to do with the graph embedding problem, as the provided data is *already* residing in a (yet high-dimensional, but still) vector space. Thus in order to reduce the dimensionality of the data, one could in principle simply apply linear or kernel Principal Component Analysis. However, in the particular case of manifold learning, the input data is assumed to lie on a low-dimensional manifold (surface) that is itself embedded in a higher dimensional space as can be seen in the example of figure 2.3. Hence, although the input data appears to be high dimensional, it varies in fact according to fewer dimensions defined by the manifold. One is then interested in retrieving

an embedding that *unfolds* the manifold into a space corresponding to its intrinsic dimensionality.

A typical method for doing so within the manifold learning literature is to build a graph-based approximation of the manifold as can be seen on the example of figure 2.3. To build such a graph, several algorithms such as the k-nearest neighbours or the ϵ-balls ones mentioned above could be used. Hence, the problem of manifold learning-based dimensionality reduction reduces to that of a graph embedding. As mentioned previously, in order to embed the graph, the manifold learning techniques then divide into local and global approaches.

Similarly to the Laplacian eigenmaps, the local methods focus mainly on preserving the vicinity of neighbour vertices. One typical such example is the *Maximum Variance Unfolding (MVU)* [116] also known as *Semidefinite Embedding* that similarly to the SPE method presented above, also relies on Semidefinite Programming for finding a better inner-product matrix K guaranteeing the proximity of neighbour nodes while at the same time pushing non-connected points away from each other. An example of global methods instead is that of *ISOMAP* [108] that simply performs a classical multidimensional scaling operation on the shortest path distance matrix of the obtained graph. The shortest distance between two points on the graph can in this case be considered as a piecewise linear approximation of the distances on the manifold, as "travelling" on the graph avoids taking any shortcuts through dimensions not belonging to the manifold as can be seen in figure 2.3. *ISOMAP* is considered as a global approach as it also takes into consideration the distances between non-neighbour nodes.

To summarize, although at first sight the problem of nonlinear dimensionality reduction appears not to have anything in common with graph embedding, a lot of effort has been invested in this field during the previous decade, and many of the well established graph embedding techniques have been revisited and updated. Hence a close observation on the future developments of this field might be in the interest of any graph embedding practitioner.

Distance matrix based methods

Another approach to the problem of graph embedding is to consider it from the point of view of distance matrices. By distance matrix, we mean the matrix D in which the entry D_{ij} for each pair of points i, j belonging to the graph G is equal to the length of the shortest

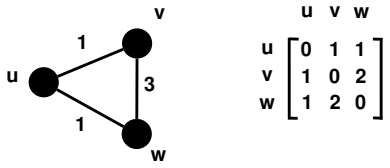


Figure 2.4: Comparison between relation (graph) view and distance (matrix) view of the data

path between i and j. Note that the undirected graph G can be either weighted, in which case the length of a path is the sum of the weights of its constituent edges, or unweighed that is equivalent to having a weight equal to 1 on all edges. Once translated into this form, one can apply a *metric embedding* method taking as input a distance matrix and finding an embedding in a given target metric space such that the distances in the embedding are in accordance with the input values. An important remark differentiating this approach from the previously presented graph drawing and spectral methods is that it focuses completely on the distance view of the data and ignores its relational property. To illustrate this, consider the example of figure 2.4 where a weighted graph comprising three vertices u, v and w and the corresponding shortest path distance matrix are shown. Notice the reported distance between the node v and w that is equal to 2. This is quite normal since the path between the two passing through u has a smaller weight sum than the direct connection. The problem when considering the data from a distance matrix point of view only is that one can no longer see the existence of the link (v, w). Indeed, one might only assume that different paths with distances larger then the reported shortest one might exist, but one cannot infer it from the distance data. Thus when considering the graph embedding from a distance matrix point of view only, one should keep in mind its lossy property. Such an issue constitutes in fact the fundamental difference between distance-vector and link-state based routing protocols in the computer networks literature.

Typical approaches of embedding an input distance matrix into a Euclidean geometric space are those of *Multi-Dimensional Scaling (MDS)* [12, 25], to which the previously presented Classical MDS belongs. Traditionally, MDS methods divide into what are called *metric* and *non-metric* methods. In the metric approaches, an exact matching of the distance values is usually desired. Note that the notion of "exact match"

relaxes to scaling operations on the distances. Indeed, given that the MDS methods developed initially as a data exploratory approach in which a researcher is interested in uncovering the structure of the elements that generated the input distances by mapping it on a visual surface (sheet of paper or screen) a scaling of the original distances by a given ratio is mainly desired. Such a scaling is also known under the name of *Ratio MDS*. In the metric MDS technique, one is mainly concerned with the minimization of an error function allowing to estimate the embedding's quality. Many such estimators exist, such as the strain function previously presented, the *raw stress*, the *S-Stress* function, *Kruskal's Stress I and II* and *Sammon's stress*. The most commonly used error measure is the *weighted stress* :

$$\sigma = \sum_{i,j} w_{ij}(d_e(i,j) - d_G(i,j))^2$$

where $d_e(i,j)$ is the distance between points i and j in the embedding (equivalent to $d_L(f(i), f(j))$), $d_G(i,j)$ is the original graph distance and w_{ij} is a weight value associated with the pair (i,j). The incorporation of such weight values in the objective function offers a lot of flexibility allowing for example to embed distance matrices with missing values as well as rectangular distance matrices indicating only the the distance between two different sets of elements (such as customers and products) by setting some of the weight values to zero in both cases. Several approaches from the nonlinear optimization literature such as gradient descent or simulated annealing can be used in order to find a solution minimizing the stress function above. A common and elegant approach based on the iterative majorization technique is proposed in [28, 46] and is known as the "Scaling by Majorizing A COmplicated Function" (*SMACOF*) algorithm. As its name indicates, this method performs the nonlinear optimization by iteratively majorizing the stress function by a simpler function for which the minimum can be analytically deduced. The algorithm then chooses the minimum of the majorizing function at iteration i as a base for the new majorizing function at iteration $i+1$. By base we mean a point at which the majorizing function's value coincides with that of the stress function while having all its other values superior to the stress values. With an increasing number of iterations, the minimum of the majorizing function and that of the stress function (or a local minimum) will coincide. Such an algorithm offers the advantage of a guaranteed convergence. We refer the interested reader to Borg and Groenen's book [12] for more details on the *SMCAOF* algorithm.

In the non-metric forms of MDS, also known as *ordinal MDS*, one is not concerned at all about preserving the original distance values (or a ratio of them), but targets mainly to preserve the order between the distances. Indeed, in many applications, it is the order between the objects of interest that matters most. Traditionally, ordinal MDS was developed in the area of psychometrics that is a sub-field of psychology focused on measuring responses to various stimuli. More precisely, psychometricians are mainly interested in uncovering the structure between the different subjects and stimuli that resulted in the observed distances. In most of the data collection settings then, the measurements did not produce a *numeric* result by attributing a value to the similarity between two stimuli A and B but rather resulted in a similarity ordering such as "stimulus A is closer to B than it is to C". Such an observation thus defines an order constraint on the distances between the mapped stimuli such that $d_G(A, B) < d_G(A, C)$. Note that one could easily attribute initial distances d_G such that they obey the order constraints and then simply use metric MDS to find a matching embedding. However, searching for an embedding that respects the constraints without tying the distances to specific values offers much more flexibility and is more in line with a data exploratory objective. Note that such an ordinal embedding is also ideal for our purpose of greedy routing. Indeed, as previously mentioned, it is in fact the order between the distances of the neighbours to the destination that matters most when choosing the next hop and not the actual values of the distances.

Ordinal MDS finds an embedding by performing an alternate optimization that is in a sense similar to the *Expectation Maximization (EM)* algorithm [32]. In a first iteration, a metric MDS embedding is performed. Note that this step requires actual distance values to be available. In case only an ordinal relationship between the distances is available, one can simply generate values respecting the order relationship. Once the metric embedding performed, the distance matrix D'_1 containing the distances between all of the embedded points at the first iteration is collected. The main idea of the alternate optimization then is to perform an optimization operation on the collected distances in order to correct them so that they follow the desired order. Note that in order for this step of the optimization to be complementary to the previous one, the searched correct distances should be the closest possible to the embedding distances D'_1. In order to perform this step, a constrained optimization technique known as *isotonic regression* is applied in which the least squared difference between the D'_1 values

and the desired corrected distances is minimized subject to the order constraints on the distances. As will be discussed in chapter 6, several order relationships can exist between the distances, some of them being *total*, meaning that a precedence relationship is indicated (or can be inferred) between any pair of nodes and some defining an order among only a subset of the distances. When a total order on the distances is defined, a simple solution to this problem consists in using the "*Pool Adjacent Violators Algorithm (PAVA)*" [72, 29] with Kruskal's *Up-and-Down-Blocks* algorithm. As its name indicates, in case a set of example constraints is defined so that $d_e(A,B) \leq d_e(A,C) \leq d_e(A,D)$ and assuming that $D'_1(A,B) = 4$, $D'_1(A,C) = 2$, and $D'_1(A,D) = 1$ then the algorithm would first proceed by *pooling* the adjacent distances $D'_1(A,B)$ and $D'_1(A,C)$ by setting them both equal to their average equal to 3. Then considering that the constraint $D'_1(A,C) \leq D'_1(A,D)$ would still be violated, the algorithm then simply sets all the three distance values to their common average i.e. $(3+3+1)/3 = 2.33$. Once the desired distances are updated so that they satisfy the order constraints, the process reiterates by performing a metric embedding (through stress minimization) of the updated distances. The process terminates when only a threshold number of constraints are violated or when the maximum number of iterations is reached.

An interesting point about the distance-based multidimensional scaling methods is that, since the search for the optimal embedding is done through the use of optimization techniques on an error function, or in other words without assuming any geometric properties on the destination space in which the graph vertices are to be embedded, these methods could in principle be applied for an embedding in any target space as long as its distance function is well defined and is differentiable. Therefore, MDS techniques for embeddings in Hilbert spaces with generic Minkowski norms (i.e. non-Euclidean) [55] as well as in hyperbolic-geometry-based spaces [26] have also been proposed.

Note that the MDS methods and in particular the alternate optimization approach can in fact reveal to be computationally very demanding and hence in some cases (especially that of ordinal MDS) applicable only to small sized datasets. A notable simplification of the MDS algorithm allowing to efficiently retrieve an MDS-like embedding is that of Landmark-MDS [30, 87]. Although this technique was mainly applied with classical MDS, it could in fact also be applied with stress-based MDS approaches. The basic idea of this approach is to select a subset of the data objects and declare them as *landmarks*. The MDS

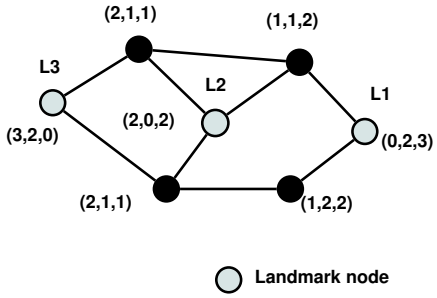


Figure 2.5: Example of a landmark-based addressing of network nodes. As shown, nodes equally distant from the landmarks would receive the same coordinates

operation is then performed based on the distances between the landmarks only, and an embedding (coordinates) for the landmark points is found. The coordinates of the non-landmark points are then computed based on linear combinations of the landmarks' positions and each node's original distances to each of the landmarks (a triangulation for example). Naturally, the success of this method highly depends on the set of chosen landmarks. This method is in fact very similar to the *Lipschitz* approaches presented below.

Lipschitz based methods

Another distance-based embedding family of methods that also relies on a subset representation of the original graph distances are the *Lipschitz* based ones [14, 61]. In this family of methods, a subset of the objects between which the distance values are reported is selected to act as a *landmark*. The coordinates of the rest of the elements are then derived from their distances to the landmarks. The main idea behind the Lipschitz embedding is that an information about the distance between any two points u and v can be extracted from their corresponding distances to the common set of landmarks. Typically, assuming that the elements for which we are provided with the distance matrix D are originally residing in a metric space, one can leverage the triangular inequality condition by selecting an element w, among the objects, as a landmark and deduce :

$$D(v,w) + D(w,u) \geq D(v,u) \tag{2.1}$$

and also that

$$D(v,u) + D(u,w) \geq D(v,w) \tag{2.2}$$

$$\Rightarrow D(v,u) \geq |D(v,w) - D(u,w)| \tag{2.3}$$

Hence, when combining (2.1) and (2.3), one can obtain both an upper and a lower bound on the distance between any two points based only on their respective distances to a third landmark point. In the classical Lipschitz method, the landmarks are in fact sets of points such that there are $(A_0, A_1, .., A_k)$ landmark sets where each A_i is itself a set of nodes. The coordinates attributed to each vertex x (in the case of the application of the Lipschitz method to graph embedding) in such a setting would the be the k-dimensional vector $(D(x, A_0), .., D(x, A_k))$ where

$$D(x, A_i) = \min_{\forall y \in A_i} D(x, y)$$

Note that the bounding rules extracted above in the case of a single landmark can be also extended to cover the case where each landmark A_i is a set of nodes. Figure 2.5 shows an example of the attributed coordinates to the graph vertices in a case where three singleton landmarks are used. An interesting result on Lipschitz embeddings is given by Linial et al. [77] that proved that when applying an L_p distance (Minkowski) metric on such obtained coordinates, one can still obtain a bounding rule such that

$$\frac{c}{\lfloor \log_2 N \rfloor} D(u,v) \leq L_p(f(u), f(v))) \leq D(u,v) \tag{2.4}$$

where $f(x)$ are the Lipschitz coordinates attributed to a node x. This means that when applying an L_p metric on the coordinates obtained through the distances to the landmarks, one is guaranteed to get a value close enough (or slightly stretched) from the original distance. We would like to point out here that such a result is in fact highly counter-intuitive. Indeed, especially when considering the case of singleton landmark sets A_i, the obtained coordinates are in fact merely a simplification of the original distance matrix D by keeping only a few of its columns (or rows) that correspond to the distances to landmarks. Such an operation

does not uncover the underlying object configuration that generated the distances, as it was the goal for example for the multi-dimensional scaling methods. Thus it might seem quite absurd to apply a Minkowski norm such as the Euclidean one on the obtained coordinates and to expect that it reflects the original distances. Nevertheless, the authors in [77] proved the above guarantee (2.4) on the distance approximation provided that an appropriate set of landmarks is selected. This is in fact the main drawback of this proposed Lipschitz embedding, as the number of required landmarks (when counting both the number of landmark sets and their cardinality) would almost be equal to the number of all elements, thus leading to very high dimensional coordinates.

Therefore, several approaches targeting to leverage the potential of Lipschitz embeddings while requiring fewer landmark elements such as [38, 45, 114] have been proposed. The *FastMap* approach [38] is in fact quite an interesting mix between landmark-based methods and (linear) dimensionality reduction techniques. The main assumption in the FastMap method is that the elements that generated the distances are residing in a high dimensional Euclidean space. The idea then is to find a way of performing a dimensionality reduction while having at hand only the distance values (this is in a sense the same problem setting as for classical MDS). The interesting property of the FastMap method is that it proceeds recursively by incrementally selecting the axis on which to perform the linear projection in order to obtain the coordinates. In order to find the first axis, the method proceeds by selecting the two most distant elements of the set, namely the two points i and j such that $D(i, j)$ is the biggest value in D. The axis on which to perform the projection is then the (imaginary) line going through the points corresponding to elements i and j in the assumed high-dimensional Euclidean space. The intention behind the selection of the two furthest elements, is to find an approximate of the axis maximizing the *variance* of the imaginary high-dimensional points similar to that identified by the PCA method. However, since we have only distance data at hand, the intuition is that the two most distant points in the dataset should reside approximately on this line. The elegance of the FastMap method resides in that it then computes the coordinates of every point according to this imaginary axis (i.e the position of its projection on the axis) by simply using *Pythagorian* rules involving the distance values of each point to the two selected landmarks, thus not requiring to know the imaginary points' coordinates nor to derive the imaginary axis' formulation. Once the coordinates according to the first axis are computed,

the method derives, again based on Pythagorean rules and the newly computed coordinates, a new distance function d_H corresponding to distances withing the $(k-1)$-dimensional subspace H orthogonal to the first axis (assuming that the imaginary space is k-dimensional). Using this distance function, the algorithm then proceeds recursively to select the next imaginary axis within H that would naturally be orthogonal to the previous one. Other similar methods such as *SparseMap* [45] and *MetricMap* [114] also perform variations on a similar mixture between landmark based methods and dimensionality reduction techniques.

In fact Lipschitz based methods bare a lot of similarity with the distance labelling techniques based on the expanding balls idea such as those of *distance oracles* and *compact routing* discussed above. Both methods rely on landmark vertices. Perhaps the main difference between the two, is that the distance labelling methods do not operate on the distance values as if they were coordinates but try to bound the unknown distances at best by storing the fewer possible values. This results in that the attributed node labels in the Lipschitz method are distance values to the selected landmarks, while in the compact routing approach they mainly consist of a combination of node and landmark identifiers.

For more details on Lipschitz methods, we refer the interested reader to the excellent review by Hjaltason and Samet [58].

2.2 Embedding techniques in the networking literature

In the networking literature, three main communities have been interested in the coordinates attribution to the participant nodes of the network. These are mainly those of *node localization*, *Round-Trip Time (RTT) estimation* and that of *greedy routing*.

In the node localization literature [36, 100, 80, 103], one is mainly interested in recovering the geographic or physical coordinate of every node. A typical application example is that of sensor networks where one is interested in identifying the location at which a measurement was taken. In a setting where only a few sensor nodes know their physical location, one is therefore interested in inferring the locations of other participant nodes from the little position information available. To the difference of greedy routing that could in principle accept any values for the coordinates as long as they lead to a successful routing, in the node localization case, there is only one ground truth for the nodes' positions

that one targets to uncover.

In the case of the RTT estimation [27, 117, 42, 105, 101], a typical application of the coordinates attribution is to be able to quickly select the closest server (in terms of transmission delay) among a list of candidates by simply checking the distances between the coordinates of the requesting node and those attributed to the candidate servers. The main motive for doing so is to avoid measuring the round-trip time to all candidates in order to determine the best one, as this would be costly in both time and resources. Hence, the main goal here is to be able to infer most of the distance values from fewer measured ones. In a sense, such an operation can be seen as a *regression* one and bares similarities with the above discussed FastMap and MetricMap methods that target to minimize the amount of distance computations, as these techniques developed within the Multimedia Retrieval community where a distance evaluation between two indexed objects can be computationally costly. In the routing community instead, the network data is usually assumed to be fully available under two principal forms, distance vectors (or matrix) or simply as a relational data (graph). Indeed, protocols from the link-state family (such as OSPF) and from the distance-vector family (RIP), as well as path-vector protocols are since long applied to real-world networks and therefore the full data collection approach is no longer a taboo in the routing community. A similar need for inference in greedy routing might be desired in highly volatile environments in which the cost of event-based updates on the global network state might be too high. In such cases, one might then be interested in inferring the new network distance values from a combination of the old values and a fewer newly measured values.

Although these three main communities have different goals and requirements, they do share the same need for (distributed) graph embedding techniques. In the rest of this chapter, we will proceed by introducing main contributions within the greedy routing literature by following the order of the embedding families described above.

2.2.1 Tree and hierarchical-based techniques

Starting with the tree-based labelling of network nodes, Kleinrock and Kamoun [65] proposed the technique of hierarchical addressing and its usage for routing. Their approach however strongly relied on routing tables and aggregation and is in fact the seminal work behind the current Internet routing methods. Tree-based greedy routing techniques have

also been more recently proposed. In the NIRA [120] proposal for example, the authors target to resolve the problem of stateless multipath routing in the inter-domain AS graph while respecting the *valley-free* forwarding rules later defined in chapter 4. They do so by assigning a unique identifier to each core domain (for example *Tier-1* domains, i.e. domains having no providers), and having each provider node attribute a descendant identifier to its customers. Routing is then performed greedily (although this was not the focus of the authors) by ascending the tree towards the least common ancestor between the source and destination that can be identified based solely on the source and destination address. In order to support multipath routing however, the proposed scheme results in a large number of attributed addresses per node.

In another recent proposal [60], the authors propose an extension to the classical hierarchical addressing scheme by embedding the network graph into a *ringed-tree* data structure that is in fact a classical tree graph in which neighbouring branch nodes are interconnected (the edges within the ringed-tree graph might in fact correspond to paths in the original graph). By attributing *polar coordinates* to the vertices of the ringed tree, a horizontal (or peripheric) greedy routing is made possible thus reducing the stretch of hierarchical routing. Such an approach is however not purely greedy as the virtual edges require the nodes to store additional control state.

An interesting extension to tree-based routing is that of *tree-covers*. In this approach, the main idea is to obtain several hierarchical addressings of the graph by labelling several different spanning trees of the same network. Following this approach, the authors in [106] propose a greedy routing method allowing to identify a path to destination along one of the trees with the least stretch. However, similar to the NIRA proposal, such a scheme induces several addresses per node as well as incurs a heavy load on the packet headers.

Another interesting usage of tree-covers is made in [40] where the authors separately embed several spanning trees of the graph into k independent d-dimensional R spaces such that the distance within each embedding is isometric to the distances on the tree. To find such an isometric embedding, the authors rely on an algorithm by Linial et al. [77]. The coordinates of the node in each of the k $\mathbb{R}^d$ spaces are concatenated to form a kd-dimensional coordinate vector. By using a modified L_∞ Minkowski norm to route greedily on the obtained coordinates, the authors prove a bounded routing stretch. The proofs are however tailored

to Unit Disk Graph models as they are the favourite tool for modelling wireless sensor networks and might not therefore easily extend to other graph families.

2.2.2 Distance labelling and compact routing techniques

Within the distance labelling approach to greedy routing on virtual coordinates, the techniques relying on hypercube methods were the first. In a 1971 paper, Graham and Pollak [54] focused on the problem of local loop switching in telephone networks, thus proposing a greedy routing methods at the very beginning of the datagram communication model. By using their graph embedding technique within a hypercube data structure [53], they attribute binary addresses to the network participants and rely on a modified hamming distance to perform greedy routing without storing any control information. Similar to the theoretical graph embedding case, the attributed node labels suffer from their length that is in the order of $O(n-1)$ bits.

In the more general approach to distance labelling, the explored techniques mainly relied on partial routing tables, making them part of the compact routing family [70, 109]. In fact one of the most interesting proposals using such approaches although pre-dating compact routing is that of *Landmark Hierarchies* by Tsuchiya [111]. In this case, each network node is defined to be a *level-0* landmark. It then announces its presence on a radius r_0 by broadcasting a HELLO message containing its unique identifier, with a TTL value of r_0 hops. Accordingly, all neighbouring nodes lying within this range will store a forwarding table entry indicating the next hop towards the level-0 landmark. A subset of the level-0 landmarks is then selected to act as level-1 landmarks and broadcast their presence on a radius $r_1 > r_0$. Restrictions on the choice of r_1's value are imposed so that a level-1 landmark is always reachable to every node with at most r_0 hops. Similarly to the previous level, all nodes receiving the level-1 HELLO message, store a forwarding entry towards the announcing landmarks. This operation can then proceed recursively until the desired number of levels is reached. The attributed label to a network node is then the hierarchical concatenation of the landmark ids at different levels from which a HELLO message was received, i.e $< Level-2\ id, Level-1\ id, Level-0\ id >$ in the case of a three-levelled hierarchy and where the Level-0 id is simply the unique id of the node. Routing to a destination label then proceeds by following

the forwarding instructions matching the lowest-level identifier in the label. Typically the packet would start by heading towards the wide-region of the destination incrementally closing in on the destination as lower-level ids are matched.

A similar and more recent example of such an approach is the *S4* proposal [78]. In this method, a set of landmark nodes is selected so that every network participant maintains a forwarding table entry (i.e. next hop) towards every landmark. In addition, each node s maintains a next hop entry for nodes within its attributed cluster C_k

$$C_k(s) = \{c \in G |\ d_G(c,s) \leq d_G(c, L(c))\}, k \geq 1$$

comprising all network nodes c belonging to the graph G, that are closer to node s than k times their distance to their closest landmark $L(c)$.

In the packet header towards a destination, one must then simply indicate the identifier of the closest landmark to the destination along with the destination's unique identifier. The packet can then be forwarded towards the closest landmark first, by using the globally stored state. When closing on the destination, the forwarding state per cluster is then leveraged to avoid passing through the landmark and to also reduce the routing stretch. Note that when $k = 1$ the proposal is in fact equivalent to compact routing in [109]. This method depends however on the number and quality of the landmarks choice.

2.2.3 Graph drawing techniques

Probably one of the most influential works in the area of geometric greedy routing is that by Rao et al known as *NoGeo* [89]. The results in this proposal were among the first to make a strong argument in favour of *virtual* geometric coordinates in sensor networks as opposed to physical coordinates. The virtual coordinates in this case are obtained through the use of a technique similar to the network of springs approach mentioned above. In fact, given that the used spring model relies simply on the elastic force without taking into account the electrostatic repulsive force between the graph vertices, such an approach can in fact be compared to a distributed multidimensional scaling operation.

In order to initiate the distributed embedding operation, nodes residing at the border (perimeter) of the sensor network must be identified. This is achieved by having one central beacon node, broadcast a hello message counting the number hops as it progresses through the network. Each node farther away from the beacon than any of its two hop

neighbourhood then declares itself as a border node. By then having all border nodes exchange hello messages, a graph distance matrix between the border nodes is derived. By applying multidimensional scaling on the obtained matrix, virtual coordinates in a low-dimensional Euclidean space are obtained for the border nodes. Non-border nodes can then be attributed initial null coordinates or random ones. The network of springs emulation begins as described above by having each node readjust its position according to that of its neighbours. An important specificity of NoGeo's network of springs approach is that the border nodes are anchored to their position (attributed through MDS) so as to avoid the collapse of all the coordinates to a single point. In a sense the NoGeo approach can also be apparented to Landmark MDS methods discussed above. The authors then demonstrate that by using Euclidean distance for routing greedily on the obtained coordinates, good routing performances are obtained. Having ourselves implemented and used this technique, we can assert its good performances on reasonably sized UDG graphs. The method however has poorer performances on other types of graphs, especially trees and scale-free graphs, as the quality of the embedding decreases.

Another approach targeting to relieve such poor performances is the *GSpring* proposal [75]. With a few differences, the virtual coordinates assignment in this proposal is pretty similar to that in NoGeo. The particularity is that the authors associate the failure of the greedy routing process (i.e. the dead-end phenomenon) from a node s to a node t with the presence of node t's coordinates within the Voronoi cell associated with s (where the Voronoi cell is computed based on the coordinates of the direct neighbours only). Therefore, in addition to the distributed embedding, the authors propose a corrective protocol in which each node *geocasts* a message destined to its associated Voronoi cell. In case an "intruder" node is detected, a repulsive force on the virtual coordinates is then exerted to move the intruder out of the cell (or equivalently move the cell away from the intruder node). At the network level, such a repulsive force can act as a "de-concavification" of the general shape of the network embedding. The greedy routing failure rate reduction remains however too low for the induced cost of the corrective protocol.

2.2.4 Lipschitz based techniques

One of the most important approaches to greedy routing when relying on the Lipschitz embedding methods is that of *Beacon Vector Routing*

(BVR) [41]. In this proposal, a given number d of landmark nodes (denoted as *beacons* in this proposal) are randomly selected and broadcast a HELLO message to the entire network. As it progresses, the HELLO message registers the number of hops traversed. The label, or in this case coordinates, attributed to each network node is then the d-dimensional vector reflecting the distances to each of the landmark nodes in accordance with the Lipschitz idea (and in contrast with the distance labelling approaches in which the attributed labels are landmark identifiers). By using a modified L_1 Minkowski distance, greedy routing proceeds on the obtained coordinates by heading towards the closest beacon to the destination as long as it is closer to the destination than to the current node, and by moving in the direction of other beacons once the current node is more in the range of the closest beacon than the destination is. To the difference of the landmark-based distance labelling techniques, the success of greedy routing is in this case not guaranteed. In some cases, the packet could end up at the closest beacon to the destination as a dead-end. In these cases a scoped flooding technique is used. A general criticism to this approach is that, similarly to the compact routing schemes, it requires a rather large number of beacons to function. This results in the inclusion of a high-dimensional coordinates vector in the packet header.

Another similar approach is the *GLIDER* proposal [39] that includes methods from compact routing to avoid routing failures by alternating between two routing modes. Initially, the network is divided into clusters, or tiles, by selecting a set of landmark nodes and the Voronoï cell associated with the node (i.e. the set of nodes closer to the landmark than to any other landmark) is declared as the landmark's cluster. Lipschitz coordinates are then attributed to each node based on its distance to nearby landmarks (after centring and squaring operations). The main specificity of the *GLIDER* proposal is that the greedy routing based on the Lipschitz coordinates is only performed within the clusters (i.e. between two nodes belonging to the same Voronoï cell). This is what the authors refer to as the local routing mode. The authors' main idea behind this, is that the difference between the Lipschitz coordinates within the cell approximate well a landmark based routing in a continuous domain that is proven to converge. For the global routing between the cells, discovery-based or simply source routing could be possible. As for all the proposals based on landmark nodes, the performances in this case also depend highly on the quality of the landmark selection.

2.2.5 Hyperbolic space techniques

An important effort in the previous years has been invested into greedy routing based on hyperbolic (or Lobachevsky) geometry coordinates. As mentioned previously, these embedding spaces present several advantages probably the most important of which for greedy routing is the exponential space expansion. This leads to an exponential growth of the distances the further we are from a reference point. Such a space reveals particularly convenient when embedding tree graphs as their number of nodes tends to grow exponentially which would lead to a compactification of the leaf vertices when embedded in a Euclidean space. Typically this would lead to greedy routing failures as the direct distance between two leaf nodes would be smaller than the distance via the parent nodes. When evaluated in the hyperbolic embedding case however, the distances between the leaves would also grow exponentially the further away from the root of the tree, thus rendering the path through the parent nodes greedily more advantageous.

Following this idea, Kleinberg [64] proposes an embedding of graphs into the hyperbolic plane and proves the success of greedy routing. Note that the embedded graph however (and that for which the routing is guaranteed), is in fact only a spanning tree of the graph instead of the complete graph. By proving that any tree graph admits a greedy embedding (i.e. an embedding guaranteeing the success of greedy routing) in the hyperbolic plane, the author deduces that by taking a spanning tree of any graph, greedy routing in the hyperbolic space is guaranteed to succeed. The author even proposes a distributed algorithm for the computation of the virtual coordinates. In [37], the authors criticize this approach, as to be fully correct, it would induce too large coordinates given the exponential increase of space, leading to exponentially growing numbers as components of the coordinates vector. The authors then propose an embedding resulting in a more *succinct* representation of coordinates using a complex data-structure they refer to as the *dyadic tree*. While acknowledging the theoretical contribution of these two methods, the proposed solutions are in our view far too complex given the resulting tree-based routing. Therefore, from a practical point of view, it might be more accessible to a practitioner to simply implement a hierarchical addressing scheme for achieving similar routing results.

In a different approach to the greedy embedding within a Hyperbolic space and focusing on the specific case of scale-free Internet-like graphs,

the authors in [84, 11, 71] leverage real-world datasets and statistical approaches such as Maximum-Likelihood Estimations to attribute the node coordinates. The main assumption being that the Internet graph growth has been "dictated" by a *hidden metric-space*, of which the authors statistically demonstrate the hyperbolic nature. Such a statistical approach to the embedding problem is a quite interesting addition to the panel of existing tools and bares a slight resemblance with the manifold learning methods discussed previously. However, the proposed approaches are so far too closely tailored to scale-free Internet-like graphs.

2.2.6 Graph sampling techniques and the revival of trees

Similar to Kleinberg's idea, other proposed approaches choose to embed a simplification of the graph such as a spanning tree instead of embedding the entire graph. The main idea being that by selecting a simpler structured sample of the network, a correct embedding can be more easily found.

In [20], the authors build a random spanning tree of the graph. Each node is then attributed n coordinates reflecting distances isometrically according to what the authors refer to as *tree-branches*. By then performing a random projection, dimensionality is reduced to a logarithmic level with a guaranteed stretch according to the Johnson-Lindenstrauss lemma [61]. Whenever a greedy routing dead-end is reached, the authors rely on tree-based routing as a recovery guaranteeing routing success with a reasonable stretch. The modalities of the isometric tree embedding are however not clear in [20].

Another recent proposal relying on spanning-tree addressing is that by Zhang et al. [121], where the authors argue against the need of a geometric, and even a metric space for greedy routing and propose an embedding of the tree vertices in a semi-metric space. In this proposal. The embedding technique is a fairly simple traversal and numbering of the tree branches and guarantees a logarithmic label size and thus a succinct namespace. By using a distance function somehow related to the L_∞ norm (by taking the minimum absolute difference instead of the maximum), the authors guarantee a successful greedy routing process. Once again, although such a result is a novel contribution to addressing and routing panel of solutions, its benefits are mainly equivalent to those of hierarchical addressing.

2.3 Summary

As seen above, a natural mismatch exists between the theoretical approaches to graph embedding and the practical takes on the problem within the networking field. One can see for example that spectral embedding methods along with Multi-Dimensional Scaling techniques have been rather snobbed by practitioners. This is most probably due to the centralized nature of these methods as they require a global knowledge as well as a centralized operation.

An important property of the Multi-Dimensional Scaling techniques however is their ability to deal with asymmetric, incomplete and multi-perception data due to their historic development within the psychometrics and social sciences fields where such data sets are frequently encountered. Such a property is therefore of high interest to us, as we later link some of the problems addressed in this work to others identified in those fields. Therefore, Multi-Dimensional Scaling embedding techniques will be playing an important role in the rest of this thesis independently from their global knowledge as well as centralized operation requirements. This is also mainly due to the fact that we target to establish a proof of concept solution to the problems addressed in the coming chapters, rather than a deployable one. Nevertheless, the proposed solutions based on centralized MDS methods could still be later-on distributed in a similar way the NoGeo proposal does distribute the embedding error minimization operation.

Chapter 3

on Guaranteeing packet delivery in greedy routing

3.1 Introduction

After more than a decade of research, geometric routing have become an attractive option for novel networks. Compared to their stateful counterparts, geometric routing protocols entail a low route-discovery overhead and are robust to topology changes. Most importantly, nodes do not need to maintain destination routing information and forward packets based only on local state. These features sum up to better scalability, energy conservation and increased robustness in face of volatile network conditions.

In geometric routing a rather simple stateless distributed algorithm routes messages from node to node along a source-destination path, based only on position information (node coordinates). A greedy routing algorithm forwards every packet to the node in the one-hop neighbourhood, which lies closest to the final destination.

In order to enable greedy routing, however, node coordinates need to reflect the relative network distances between nodes. As the assignment of node coordinates might not be a perfect approximation of network distances or due to the presence of obstacles, in practice, simple greedy routing is not always guaranteed to work. Depending on the geometry of the resulting network graph, the density of the network and the presence of void regions in the graph, the forwarding process might halt at local minima nodes with no neighbour closer to the destination than themselves.

In this chapter, we explore the effects of clustering as a low cost approach for improving the performance of greedy routing. The key idea is to re-use the connectivity information acquired during the preprocessing phase, and construct a graph of interconnected clusters that form a reduced connectivity topology. Node-local routing decisions thereafter, by means of the same greedy algorithm, can be performed either at the *base level* (based on actual node coordinates) or at the cluster level (based on cluster coordinates). The assumption we make and empirically evaluate in this work is that local minima and void regions in the connectivity graph of a sensor field (base level) are likely to be transposed or reformed at the cluster level view, which when combined with routing at the base level graph are often overcome. Our aim is therefore to measure to which extend such additional low-cost embeddings would improve the greedy routing performance compared to a single high-dimensional one. As the work in chapter can be seen as an extension to the classical greedy routing technique, a main concern

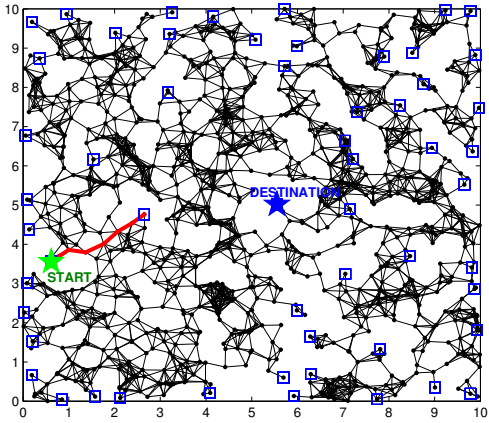
here is then to be compliant with al types of networks, including mainly sensor networks, as they make at most use of greedy routing.

The remaining of this chapter is organised as follows. We start by describing the problem of greedy routing dead-ends due to local minima in section 3.2. We then describe the various approaches in literature in section 3.3. Section 3.4 provides a description of our framework for deploying clustering and constructing hierarchical cluster views (levels) for greedy routing. In Section 3.5 we provide an evaluation over multiple configurations, of clustering methods, coordinate embeddings, and topologies, reporting a significant improvement in the effectiveness of greedy routing. These results essentially ratify our vote for simplicity. Section 3.6 provides some general insightful conclusions on our observations, and finally section 3.7 summarises and concludes the paper.

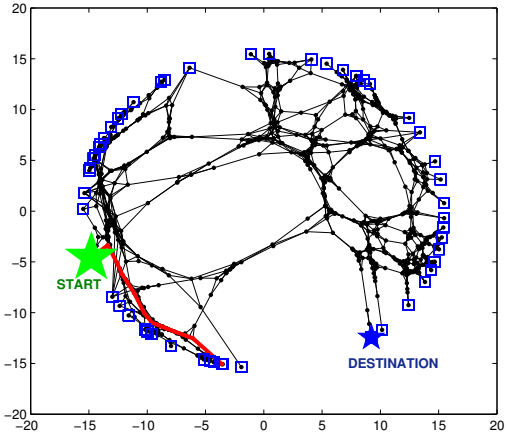
3.2 Greedy Routing Dead-End Problem

As previously detailed about the greedy routing algorithm, an important condition for the loop avoidance of the routing procedure is that a node forwards a packet to a neighbour **only if** the neighbour is closer to the destination than the current node. Therefore, in case the computed virtual coordinates do not correctly reflect the order of distances between all nodes, a situation where a node finds itself closer to the destination than all of its neighbours might frequently happen. Figure 3.2 shows several examples of such a situation according to different embedding methods. In this figure, an example network of 1920 nodes was generated using the unit-disk graph generation model, in which network nodes are randomly spread around a surface, and two nodes are considered to be connected if they lie within a threshold distance from each other. This network with the "physical coordinates" is displayed in figure 3.1a along with an example of a failed greedy route based on Euclidean distances on the physical coordinates. As can be seen, the node at which the routing failure occurs is closer to destination than all of its neighbours due to the presence of a void area between it and the destination. More precisely the greedy routing failure is due to the shape of the face (i.e plane area) enclosed by the void that presents a concavity at the failing node. In case the void region would have been convex at the failed node, a neighbour closer the destination could always be found.

In fact, greedy routing performs better when based on *virtual* coordi-

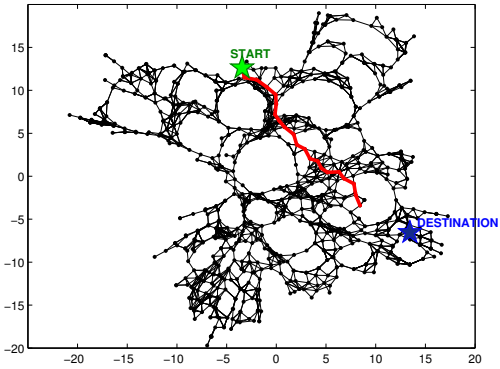


(a) Generated network using unit-disk graph method with 1920 nodes

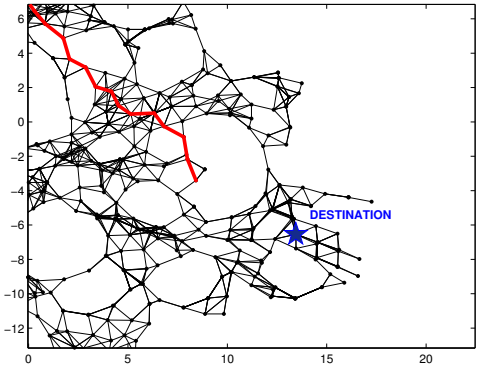


(b) NoGeo 2D Embedding and example greedy failure (Embedding stress 0.3125)

Figure 3.1: Various greedy routing dead-end examples



(a) SMACOF 2D embedding and example greedy failure (Embedding stress 0.119)



(b) Zoom on the SMACOF dead-end case. One can see that the slightest distortion in the embedding can lead to a dead-end, although globally the path was valid

Figure 3.2: Various greedy routing dead-end examples

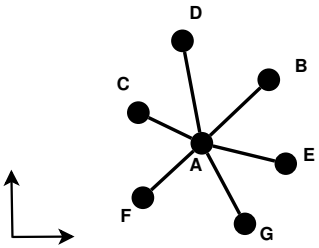


Figure 3.3: Example of greedily unembeddable graph in 2D Euclidean space

nates as the graph embedding techniques usually produce embeddings with (mostly) convex void regions as can be seen in figures 3.1b and 3.2a. In figure 3.1b, an embedding of the graph in a two-dimensional Euclidean space is produced using the NoGeo [89] technique. The border nodes used for this embedding are those indicated by the blue squared nodes in figure 3.1a. As can be seen from this embedding figure, the NoGeo method forces the virtual coordinates to adopt a (excessively) convex shape. This however does not solve all the greedy routing dead-end problems as the network seems to fold on itself, thus producing a concave external face (the white space surrounding the graph). In figure 3.2a a better embedding is obtained when using the metric multidimensional scaling embedding technique via the SMACOF algorithm [12]. However, as can be seen from the example failed greedy route in this figure and in figure 3.2b where a closer look at the failure point is shown, the slightest misrepresentation of the network distances might lead to a greedy routing dead-end, even if the route to the destination was globally on the right path.

Such cases of greedy routing failure are often due to distortions between the embedding space distances and the real network distances. Indeed as discussed in the previous chapter, an exact representation of the network distances in a low-dimensional Euclidean space is in most cases practically infeasible. This leads to situations as in the SMACOF embedding case of figure 3.2a (in which the Kruskal embedding stress I is 0.119 i.e. reasonably low) where a slight deviation of the embedding distances can ruin a globally satisfactory result. Therefore, one obvious measure against the greedy routing dead-ends might consist in obtaining better quality embeddings that better preserve the order between the original network distances. This can be achieved either by adopting a better embedding techniques, as it was the case for our example

network in which the centralized stress minimization SMACOF method performs better than the decentralized NoGeo, or, in case the embedding method is fixed, by increasing the dimensionality of the target embedding space. Indeed adding more dimensions to the target space offers more room and thus possibilities of placing the network nodes differently, hence resulting in a decrease in the embedding stress (or distortion) and therefore in a decrease of greedy routing failures.

Note however than in some cases, no matter the quality of the embedding, greedy routing failures are deemed to be unavoidable. This is due to the fact that certain families of graphs cannot possibly be embedded in some target space so that a perfect greedy routing, where all routes between all pairs of points are correct, is possible. This has for example been demonstrated by Papadimitriou et al [83], that have shown that the bipartite graph $K_{1,6}$ (star graph with one centre node and six neighbours) and its generalization, the family of bipartite graphs $K_{k,k+5}$ do not admit an embedding in the two-dimensional Euclidean space allowing for a successful greedy route between all pairs of points. The example case of the star graph is shown in figure 3.3. In the embedding of the graph in the figure, one can see that no greedy route is possible from node E to G as E is closer to the destination than its neighbour A. An intuitive demonstration of the infeasibility of an embedding guaranteeing successful greedy routing in this case, is that in order for the routes between any two nodes X, Y different from A, to pass through node A in both directions, then the line segment $[X, Y]$ must be the hypotenuse of the triangle XAY hence inducing that the angle $\angle XAY$ must be superior to $\frac{\pi}{3}$. This therefore limits the possible neighbours of A to five as the sum of all adjacent $\angle XAY$ angles must be inferior or equal to 2π.

3.3 Related work

Early work on geometric routing was focused on physical geographic coordinates (e.g. [63, 13, 73] and other), which fuelled research on localisation and approximation of the actual network node coordinates from partial geo-positioning information [100, 36, 15, 105]. As node coordinates involved geo-positioning, addressing routing problems related to the morphology of the coordinate space (presence of void areas) led to hybrid routing approaches alternating between the classical greedy routing mode and another *recovery mode* that is enabled whenever a greedy

routing dead-end is met. One example of such a recovery method is *scoped flooding* that is notably used in [89] where the network node at which the greedy routing dead-end occurred simply broadcasts (floods) the packet to all of its direct neighbours that in turn are requested to broadcast it to all their neighbours until the number of broadcast hops reaches a fixed threshold (hence the *scoped* denomination). The hope is that one of the flooded packets would be able exit the void's concavity and reach a point where a greedy route towards the destination is again possible. This method however offers no guarantee that the packet would not, after a few greedy steps, fall again into the same local-minimum and poses additionally the problem of packet replicas, as for each greedy dead-end encountered towards the destination, multiple replicas of the packet are created that in turn can generate other replicas in dead-end situations.

Another one such recovery mode is the quite famous technique of *face routing* [13, 63, 43, 74]. In this particular recovery mode, the connectivity graph is modified (by ignoring some of the links) in order to obtain a planar graph, i.e a graph drawn on a euclidean two-dimensional plane so that none of its edges intersect in points other than the graph vertices. This graph planarization is usually achieved via distributed algorithms computing variants on *Gabriel Graphs* and *Delaunay triangulations*. Whenever a greedy routing failure is encountered, face-routing then proceeds by routing along the interior of the faces of the planar graph (by following the *right hand rule*) and switching between faces only when a face closer to the destination node is encountered. In the most classical implementation, the distance of the face to the destination is determined by the distance of the point of intersection of one of the face edges with the straight line between the source and the destination coordinates. Note that when given a planar-graph, face-routing methods could be applied on their own as a routing technique, independently from greedy geometric routing. Their major disadvantage then being a rather large worst case routing stretch, in addition to the requirement of a planar graph.

While geographic positioning coordinates provide a natural metric space for geometric routing, *virtual coordinates* offer an alternative whereby the coordinates assigned to nodes are not implied from physical locations but can be *chosen* to fit certain criteria. Such criteria typically include the shape and dimensionality of the coordinate space, the convexity of void regions, and other, which aim to enhance the performance of greedy routing itself without resorting to face routing.

The use of solely virtual coordinates for greedy geometric routing became popular with the *NoGeo*[89] proposal. Due to its elegant simplicity and good performance, this seminal work has been used thereafter as a reference system for comparisons. At the same time it also fuelled further research on understanding under what conditions (topology of the graph, adjustments of the routing metric), low-dimensional graph embeddings on the plane, guarantee successful greedy routing [83, 34, 5]. A general insight in these works is that not every network graph admits to a Euclidean embedding that guarantees greedy routing as previously discussed.

A notable example of a virtual coordinates method where a better embedding is searched is GSpring [75] discussed in the previous chapter. In this approach the virtual coordinates are initialized according to a model slightly different from the one in [89] but still in the framework of network or springs graph drawing methods. The specificity of the GSpring approach however is that the embedded network nodes continuously try to better the quality of the embedding by detecting and correcting greedy routing failures. This method however may suffer from oscillation issues, as different nodes' corrections can be conflicting thus resulting in a perpetual oscillating coordinates. Moreover, the correction mechanism in this approach is dependant on the problem to be corrected. Indeed, in order to correct the embedding for a better functioning greedy routing, this approach relies on greedy routing to perform the geocast operations, hence no correction could be possible in the case of a highly dysfunctional routing.

In the proposal by Kleinberg [64], discussed in the previous chapter, the author proposed to embed network graphs into the hyperbolic space and proved that any graph admits an embedding guaranteeing the success of greedy routing between all pairs of nodes. He also proposes a distributed method for computing the node coordinates in the hyperbolic plane. However, and as pointed by [37], the major drawback of using hyperbolic virtual coordinates is that the coordinate values would also increase exponentially, thus requiring an addressing scheme with $\Omega(n \log(n))$ bits in the worst case to encode the hyperbolic virtual coordinates. In their paper [37] Eppstein and Goodrich propose a more *succint* representation of the virtual coordinates requiring only $\Omega(\log(n))$ bits for every coordinate through the use of a complex data structure denoted as *dyadic tree*.

Although these two methods brought significant theoretical results to the field of greedy geometric routing, they do not seem to be picked

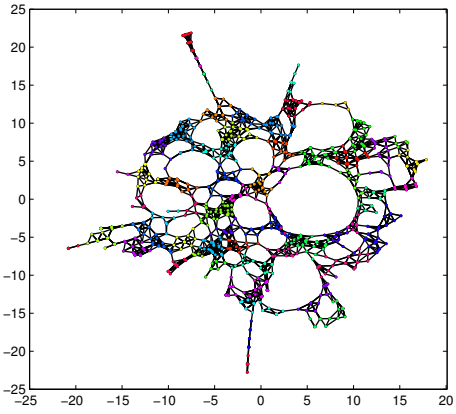
by the practician community. This is not only related to the theoretical complexity of the proposed methods, but also to the fact that they both guarantee in reality a greedy routing following paths along the spanning trees of network graphs. This is to say that, in practice, a more classical approach of attributing hierarchical addresses (such as IP addresses) to the network nodes along the spanning tree, and relying on hierarchical tree distance (or longest prefix-match) method would achieve the same goal. Indeed, provided that the hierarchical addresses are correctly attributed, such a greedy routing technique would guarantee a successful route between all pairs of nodes without requiring any additional state than the list of neighbour nodes. While acknowledging the significance and theoretical foundations of these contributions, in this chapter we chose to empirically explore if simple and computationally more sustainable tactics that lend to immediate deployment with different types of networks including small sensors, can yield comparable effects (i.e. overcome local minima, nullify void regions). As we show later, our first results support this intuition.

Finally a work, which bears a notable relevance to the one presented here, is presented in [39]. The authors exploit a clustering method to divide a network in compartments (*tiles*), based on which they factor the routing process in inter- and subsequently intra-compartment routing. The clustering method creates a *Voronoi complex* around a set of appointed landmarks (cluster heads), which then function as "attractors" for the different segments of routing paths. A cluster graph is the result of a *Delauney triangulation* among the landmarks.

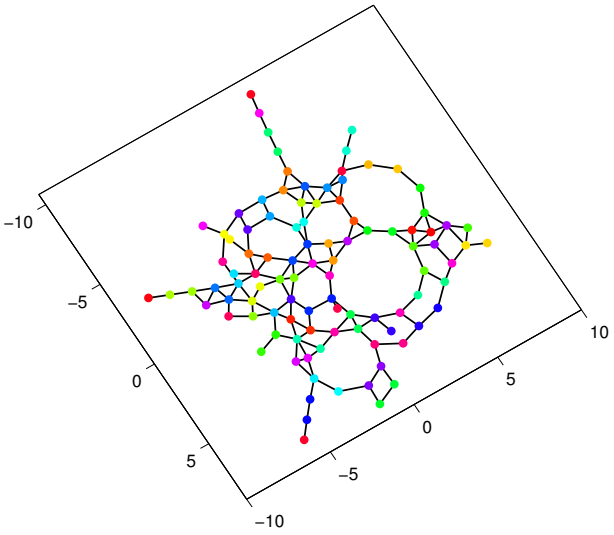
Our work goes beyond a specific cluster mechanism and is more broadly scoped on the effects of clustering. From this end, we observed that it is actually more effective to start routing at the base level and resort to the cluster level only when local minima are found, by contrast to the strategy in [39] where routing always starts at the tile-level. Another realisation from our exploration is the un-necessitated need for landmarks to achieve the path-curving effects of [39]. Finally we do not resort to explicit planarisation of the embedded coordinate space.

3.4 A cluster-based approach

In cases of an already deployed geometric routing system, the only possible corrective measure that could improve the quality of the greedy embedding would be to increase the dimensionality of the target geo-



(a) MDS Embedding of network with 1000 nodes and result of agglomerative clustering with 100 obtained clusters



(b) MDS Embedding of the resulting cluster graph (vertex colours correspond to the cluster colour in (a))

Figure 3.4: Example of detail-level views of the network coordinates

metric space. This would reduce the distortions between the embedded and network distances and would therefore reduce the greedy routing errors.

Our goal in the rest of this chapter is to evaluate the feasibility of another, non-explored, embedding enhancement method in such scenarios, when relying on clustering. Our intuition is that by routing greedily on a *coarser* view of the network, the participating nodes can take decisions based on regional proximities allowing to oversee small disparities between the embedded and the network distances. Said differently, picturing the network graph at the cluster level should allow to blur away small void concavities at the lower network level similar to the one in figure 3.2b. An example of this intuition is pictured in figure 3.4, where a multi-dimensional scaling embedding of a 1000 nodes network is first shown in figure 3.4a along with the result of an agglomerative clustering procedure. In figure 3.4b, the embedding with the same method of the resulting cluster graph (in which two clusters are neighbours if two of their nodes are neighbours at the flat level). As can be observed from the figure, the same structure of the network graph is extracted after the embedding, however stripped out of low-level details. Our intention therefore is to exploit the different details capability offered by the combined clustering and embedding in order to obtain better greedy routing results.

We will experimentally demonstrate in the following that relying on the clustering approaches, for bettering the embedding qualities, achieves better routing results than the dimensionality increase approach, at a similar cost.

To test our ideas and hypotheses we developed a simple framework that enabled us to experiment with different clustering algorithms and embedding approaches in Matlab. The main components of this framework, which are detailed in the following sections are responsible for the creation of clusters, their embedding in a virtual coordinate space, the management of routing state, and finally for applying a greedy routing strategy based on the selected distance metric.

3.4.1 Construction of a cluster graph

Our method requires a cluster graph view of the network. Most of the proposed virtual coordinate schemes rely on a global preprocessing phase to discover the network connectivity, and thereafter a local routing phase where a greedy algorithm is used to forward messages across the

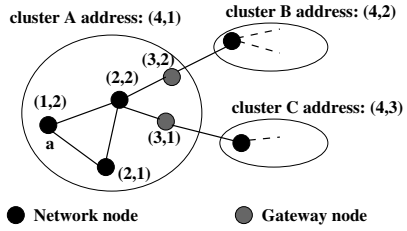
network. During the initial phase a cluster graph can be produced by assigning the network graph nodes to groups (clusters) and then consider the adjacency relations between the different formed clusters.

One can use any of the graph clustering techniques available in literature [98], such as spectral clustering, minimum-cut based approaches, agglomerative methods, or other. However, a key requirement for efficient application scenarios such as wireless sensor networks, is that the clustering algorithm is simple enough for fast and distributed computation. To this end, clustering protocols from the wireless sensor networks literature such as [76, 6] can be applied. Naturally the quality of a clustering method in place, will affect the intra-cluster density as well as the cluster graph connectivity, and therefore is likely to influence the performance of greedy routing. As our goal is to study the average qualitative effects of clustering, rather than to seek a highly effective clustering algorithm, we experiment with two very different algorithms (of diverse sophistication) and multiple seeded configurations. This enabled us to produce a broad variety of cluster graphs for the evaluation. The algorithms we used were a spectral clustering variant of [51, 22], and at the other end of the range we employed a simple agglomerative approach, both of which are discussed further in the evaluation section.

3.4.2 Cluster graph embedding

In the second step, we need to embed the cluster graph so as to be able to perform greedy routing on it. Here, it would be possible to produce a new embedding into any metric space (not necessarily Euclidean) and of any dimensionality. However proposing a new embedding is beyond our scope as it would bias our objective of assessing the effects of clustering over existing systems. For this reason, we apply the same embedding technique that is being used at the base level. At the same time, as we discuss later on, this enables multi-level scaling and is aligned with our second goal to keep the deployment cost minimal by re-using the existing algorithmic code-base.

Our reference systems have been NoGeo, classical MDS and metric MDS and therefore we resort to the same embedding techniques to produce the embedding of the cluster graph in a d-dimensional Euclidean space (d being equal to the dimensionality of the base graph embedding). For more information on these systems we refer the reader to chapter 2 as well as [89, 12]. At the end of this process each cluster receives a coordinate in a metric space, such that the same greedy routing



(a) Network and cluster addressing

Flat Level Neighbour Table

(2,2)
(2,1)

Cluster Level Neighbour Table

Cluster	Gateway
(4,2)	(3,2)
(4,3)	(3,1)

(b) Neighborhood information at node a

Figure 3.5: Network and cluster Graph embedding example

algorithm can be performed interchangeably on the cluster graph and the node graph.

3.4.3 Cluster-level state

In our framework we do not rely on cluster heads. Instead all members of a cluster maintain state that allows them to share the same knowledge about the cluster. This avoids the operational dependency on particular nodes as well as the need for additional election algorithms. Specifically, each node maintains the coordinates of its cluster in the embedding of the cluster graph, and the cluster adjacency information of neighbour clusters. The latter consists of the neighbour cluster coordinates alongside a respective inter-cluster gateway's coordinates. An inter-cluster gateway is any network node in a cluster that has at least one neighbour in another cluster.

Figure 3.5 exemplifies the overall neighbourhood information stored at each node for the base level and the additional cluster level. Note that the cluster-level information should not impose any scalability problems as it would sum up to $O(k)$ (k being the number of neighbour clusters – independent of the network size, and related to the density of the graph). An evaluation of the memory cost entailed is provided in Section 3.5.4.

3.4.4 Operation of greedy routing

In order to enable greedy routing at the cluster level, the packet header has now been extended so as to contain both the destination node's coordinates in the base level embedding, as well as those of its cluster in the embedding of the cluster graph.

Routing at the cluster level

Given a destination cluster coordinates a forwarding node first computes the distance from its own cluster. It then selects among its neighbour clusters the one with the smallest distance to the destination's cluster. If the current cluster is closer to destination than the selected neighbour cluster, a stop condition is reached at the cluster level (in accordance with the basic greedy algorithm). In order to reach a selected neighbour cluster, a node needs first to route greedily to the respective gateway node in its own cluster. Hence the gateway's coordinates are prepended in the packet that is greedily forwarded to it. Finally, at the neighbour cluster's gateway node the temporary coordinates are removed and forwarding towards the destination coordinates is resumed.

Note that a stop condition might also be reached in case the neighbour cluster's gateway is not greedily reachable. Such a phenomenon may occur if the clustering approach introduces concave void regions in the intra-cluster graph.

Cross-level routing

The benefits of clustering come into effect when one combines greedy routing at the cluster level with routing at the base level. We are currently considering two possibilities, in the first, denoted as top-down, greedy routing starts at the cluster-level and concludes at the base level, while in the other, denoted bottom-up, routing starts at the base level and, if a local minimum is encountered, the algorithm tries to progress further at the cluster level.

In the top-down approach, even if the destination cluster is reached, routing has to always switch to the base level for intra-cluster delivery to the destination node. By contrast in the bottom-up approach, it is likely that routing is concluded at the base level (if no local minima are encountered) without switching to the cluster level graph.

A third possibility would be to continually alternate between the two levels, whenever a local minimum is encountered. However, unless some

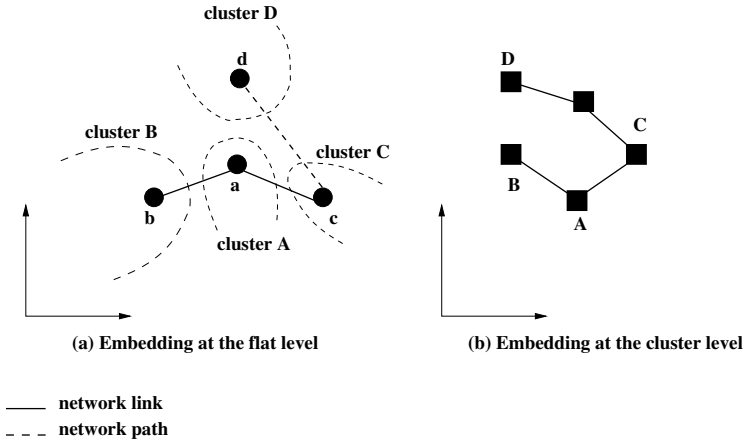


Figure 3.6: Greedy routing loops between network levels

additional sophistication is introduced (constraints), this behaviour can lead to routing loops as exemplified in Figure 3.6. Node a which is a local minimum at the base level (Figure 3.6.a), wishes to greedily route to node d. By switching to the cluster level and using the cluster level embedding of Figure 3.6.b, greedy progression is possible and node a forwards the packet to cluster B, i.e. at gateway node b. However, cluster B is a local minimum at the cluster graph, and routing at this level reaches a stop condition. Switching back to the base level, will result in b forwarding the packet back to a, and an infinite loop is created. Clustering in this case will not solve the problem. To avoid worsening the problem either, for now we forbid the switching between the two levels more than twice, by nullifying the destination cluster's address in the packet.

3.4.5 Multiple cluster levels

The practise that we have described so far can be repeated to introduce additional cluster levels, by recursively clustering over the cluster graph. The greedy routing would function in a similar way to the two-level case, however, the gateway nodes between clusters at the third level are in fact clusters as well (of level 2).

The rationale is that while the first cluster level performs some degree of "averaging" over the network graph connectivity and "smooth-

ing" the concave voids, the additional clustering further refines the abstract topology by averaging and smoothing among cluster groups. This would effect a more gradual (as opposed to sudden) bending of the (Euclidean) path between two nodes, however at the cost of higher *path stretch*. In our evaluation we experimented with up to two levels of clustering, in order to discover the degree of additional improvement compared to the cost of this process.

Another approach when experimenting with multiple cluster levels would be to use more than one cluster graphs in parallel, at the same level (each produced with different clustering method or the same method with a different seed). Using a copy and branch operation during the switch to the cluster level, greedy routing would then be carried out in both cluster graphs. The idea would be that while routing along one cluster graph might reach a stop condition, it may progress along the other. In case the packet gets delivered along both paths the destination would eliminate the copies. This approach however requires further investigation in regard to choosing the right action policy when stop conditions are met. We have not experimented yet towards this direction.

3.5 Evaluation

In this section we present the results of exploration with clustering. We report on the performance improvement observed (in finding greedy paths) compared to three reference geometric embeddings, namely No-Geo, classical MDS and metric MDS. We also examine the overhead of clustering regarding the additional per-node state and the average per-path routing stretch.

3.5.1 Experimental set-up

All simulated topologies have been produced using the Unit Disk Graph approach [23], by randomly scattering nodes on a fixed size grid and establishing a link between two nodes whenever a minimum threshold distance exists between them. The number of nodes in the topologies range from 500 to 5000, incremented in 500 nodes steps. The tests are therefore repeated as a function of network size, where we keep the graph density constant: average node degree 7 (this degree exhibits a reasonable number of route failures in the base embeddings to allow for comparison). We also conducted tests as a function of the network

density, where we keep the network size fixed at 2000 nodes and vary the average node degree (by changing the Unit Disk radius).

The two MDS systems were based on the Matlab implementations, while the NoGeo system was based on a simulator running the network of springs method described in [89].

3.5.2 Clustering algorithm used for the tests

As we are interested in exploring the effects of clustering in general rather than focusing on a specific clustering approach, we have used two diametrically opposite clustering methods in terms of sophistication. This should provide us with some boundary insights on the performance variability in regard to clustering method sophistication.

The first is a centralised spectral-clustering algorithm [51, 22] that tries to find optimal clusters following a relaxation of the normalized minimal cut problem through an eigen-decomposition of the Laplacian matrix discussed in the previous chapter. This method relies on knowledge of the graph's adjacency matrix.The second is a simple distributed agglomerative clustering method that creates clusters in a stochastic manner. First, a "root" node initiates the cluster formation by broadcasting a *cluster-join* to its k-hop neighbours (where k is set in advance). Nodes that are not already members of a cluster, upon receiving the invitation will join the cluster. Then a new root node is chosen in the proximity of the last formed cluster, and the operation repeats until all nodes are assigned to a cluster.

In all tests the generated cluster graph is embedded in a 3-dimensional Euclidean space.

3.5.3 Greedy routing performance

To evaluate the routing performance, we forward greedily a packet from every node in the network to all other nodes ($N \times (N-1)$ routes for a network of N nodes), and we monitor the percentage of those packets that failed to reach their destination. We call this metric *Greedy Failure Rate (GFR)*, and we plot it as a function of network size and network density. The results shown in the figures are averages over 10 runs with topologies of same characteristics (network size and density).

One first general observation regarding the cross-level routing, confirmed in all plots, is that the bottom-up strategy always yields better

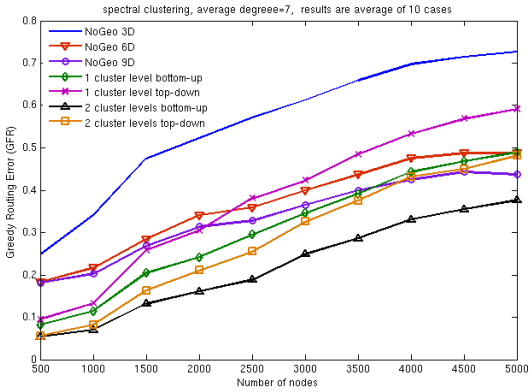


Figure 3.7: NoGeo greedy route failures with spectral clustering

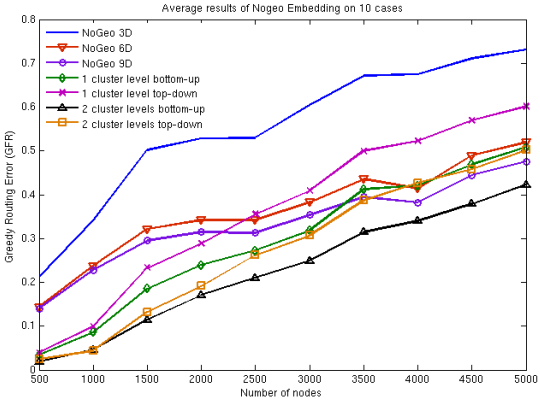


Figure 3.8: NoGeo greedy route failures with agglomerative clustering

results than the top-down approach. This validates our intuition explained in section 3.4.4.

Starting with the NoGeo system, Figure 3.7 reports the GFR as a function of the network size, when the NoGeo embedding was used in combination with spectral clustering. Figure 3.8 reports on the same test when the agglomerative clustering was used instead. In each of them, we practically evaluated two orthogonal aspects: one being the

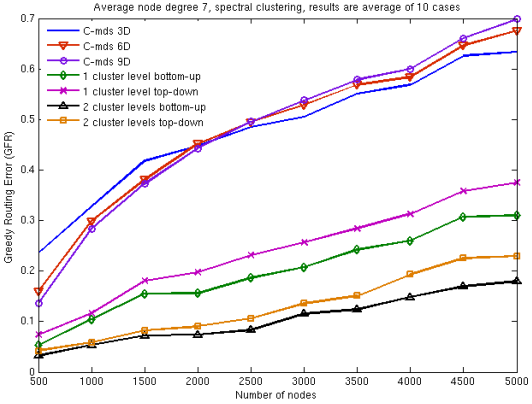


Figure 3.9: Classical MDS embedding using spectral clustering

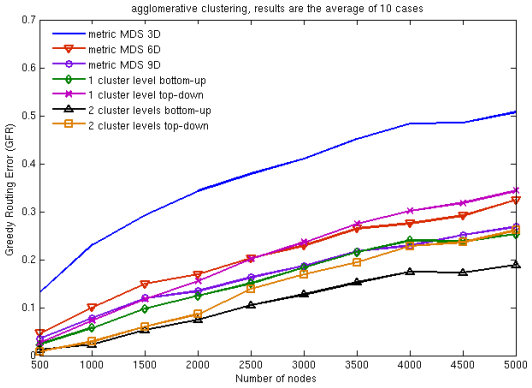


Figure 3.10: Metric MDS Greedy routes failure using agglomerative clustering

absolute performance improvement (GFR reduction) when we employ clustering over the base embedding, and the second is how this improvement compares to the performance of a more accurate base embedding in a higher dimensional space (3D, 6D and 9D curves in the figures).

One sees a notable improvement from the introduction of clustering, which for the various network sizes, ranges between 15% and 25%

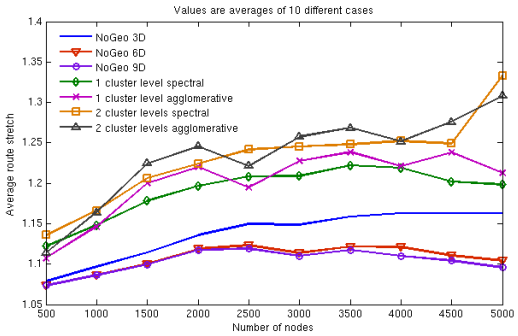


Figure 3.11: Average stretch using NoGeo embedding

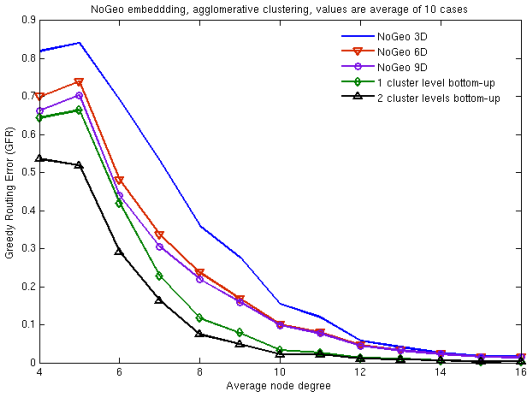


Figure 3.12: Average node degree variations on a 2000 nodes network

when one cluster level was used, and between 20% and 35% when two cluster levels were used. An interesting observation is that between agglomerative and spectral clustering there is no substantial difference in the average GFR, which suggests that the measured effects might be statistically quantifiable for different embedding systems, irrespective of clustering method. However, such a hypothesis needs more thorough examination with additional clustering methods. The variance in the datasets (may be perceived in the smoother curves) for the GFR, was smaller in the case of spectral clustering, most likely because of its less

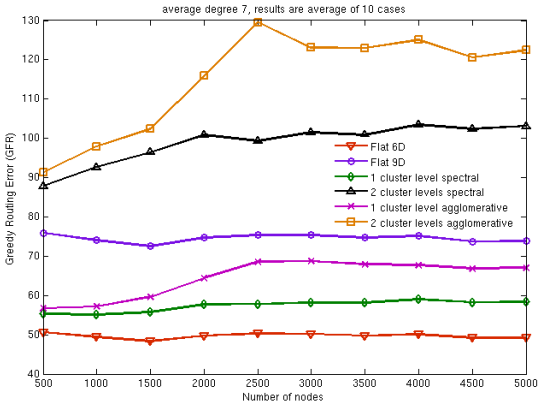


Figure 3.13: Average storage overhead per node for 1 and 2 cluster levels

stochastic nature (in generating cluster graphs).

Looking at the groups of curves in the same two figures, one can observe an asymptotic reduction of the GFR as the dimensionality of the base embedding (NoGeo) increases from 3D to 9D, and which is further "magnified" in larger networks. An analogous pattern seems to exist as the number of cluster levels increases in all configurations. A noticeable difference, between these two groups, is that the NoGeo lines are curved with a monotonically decreasing angle, compared to the more linear clustering performance curves. As a result although clustering yields better results than high dimensional embeddings at the beginning, as the network size increases (with a fixed density), a high-dimensional embedding scales better.

Analogous improvements and observations are reported in Figures 3.9 and 3.10, in comparison to the classical MDS and metric MDS base embeddings respectively. As the results were similar in all configurations, we only show metric MDS with agglomerative clustering, and classical MDS with spectral clustering. An additional observation here, in the case of classical MDS (Figure 3.9), is that clustering performs substantially better. On the other hand, increasing the dimensionality of the base embedding does not noticeably improve the performance, rather after a certain network size it actually worsens it.

Looking at the consequent average route stretch factor (Figure 3.11) of the successful greedy routes, as anticipated (and conformed also in

[39]) the different clustering methods, transparently from the greedy algorithm, curve the Euclidean paths in order to get round the local minima. Yet, the increase of the stretch factor is quite small, i.e. ≤ 0.15 of the respective stretch of the 6D and 9D NoGeo, and ≤ 0.09 of the 3D NoGeo.

Regarding the network density, Figure 3.12 reports on the performance of the same configurations as before, as a function of the network density in the case of a NoGeo embedding. Clustering in this case provides a significant performance improvement in networks with an average degree below *11*. Beyond this threshold, clustering still performs better, however with a minimal improvement, as the performance of all configurations converges asymptotically to near-zero GFR.

3.5.4 State overhead for clustering

To assess the overheads introduced by using our framework for supporting clustering in geometric routing, we assume the following simple reference model for comparisons: For addressing across l hierarchically organised cluster levels, each embedded in a d dimensional space, a node's addressing information occupies $l \times d$ units in the packet header. This is required to enable greedy routing decisions and distance calculations at all different levels (1 unit comprising the space requirement for a coordinate, in one dimension).

To increase on the other hand the performance of greedy routing using an embedding approach (without clustering), one can increase the dimensionality of the embedding space. An increasing of the dimensionality to $d' = (l \times d)$, would impose the same addressing space requirements as l clusters (same packet overhead, and each neighbour record in the adjacency matrix of a node, would occupy the same amount of space). This analogy will be the basis for our memory overheads comparisons (multiple clustering levels versus a high dimensional base embedding). It practically means that we would like to compare 1-level clustering regime with a 6D base embedding of the network graph, and 2-level clustering with a 9D base embedding of the network graph.

We then calculate the space overheads of the different clustering configurations evaluated earlier as follows: For a d-dimensional base embedding of a network graph with approximate density (node degree) k, the average amount of adjacency state maintained in each node is $(k+1)\times d$. In the case of clustering with our framework, using l cluster levels, the same d-dimensional embedding across all levels, and an average

node degree k_i for the graph in level l_i, the amount of adjacency state maintained in each node totals $l \times d + \sum_{i=1}^{l}(2 \times k_i \times d)$ (multiplier 2 represents the fact that for each cluster adjacency record a node needs to store both the cluster address and a respective gateway address).

Figure 3.13 reports the resulting averaged estimates over the 10 topologies, for each of the configurations discussed in the previous section. In all cases but one, the memory cost does not vary significantly as the network size increases. The exception were the configurations where agglomerative clustering was used. The exception of agglomerative clustering can be explained by the variability in the clustering performance for the agglomerative methods, which leads to cluster graphs with different average node degrees. This result suggests that the choice of clustering method has a variable impact, but a careful choice of clustering method can retain the scalability in regard to network sizes.

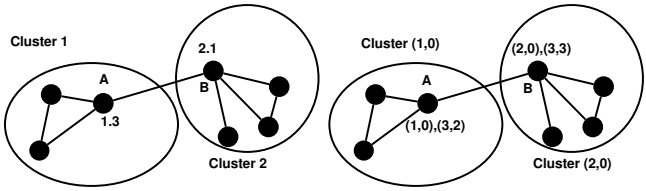
A comparative examination of the plots reveals that the overhead of one level of spectral clustering when compared to a 6D embedding would amount to having each node store less than two additional 6D neighbour addresses (10 additional unit coordinates). In the case of a 2-leveled spectral clustering compared to a 9D embedding, the overhead amounts to storing three additional 9D addresses. On average, one sees an overhead of approximately 8-15% in the case of spectral clustering and 15-35% in the case of agglomerative clustering. This overhead becomes 15-35% and 20-70% respectively, when comparing 2-level clustering with the 9-dimensional base embedding. At the same time the cost from a 6-dimensional base embedding to a 9-dimensional one increased by 50%. Therefore, roughly we may say that the overhead of adding clustering levels increases twice as fast as the cost of the corresponding dimensionality increase in the base embedding.

3.6 Discussion

3.6.1 On the general effects of clustering

Clustering in greedy geometric routing has the potential of indirectly creating two interesting effects, which motivated our work. Both of these effects are confirmed by our results.

The first is on void regions that are the product of an embedding of the network graph in a low dimensional space. On one hand the cluster graph essentially "thins" the densely connected regions of the network graph, which results in a more uniformly connected and spread-out sub-



(a) Classical hierarchical addressing

(b) Our coordinates hierearchies

Figure 3.14: Difference from classical hierarchical addressing

graph. Thinning happens as the clustering process "packs" these regions inside cluster nodes. As a result of this the routing process is factored in two levels: (a) within clusters where the dense connectivity is more likely to yield successful routing under a greedy regime, and (b) across clusters whose more uniform connectivity and arrangement in their coordinate space reduces areas of local minima and makes greedy routing more effective. From a macroscopic perspective the overall effect can be seen as a thinning and zooming out process that dilutes void regions.

On the other hand clusters function as attractors that curve the trajectory of routing paths. To understand how this works imagine that in greedy routing the destination applies a pulling force on the packet, which in every forwarding decision pulls the packet a little bit closer. Then every time routing decisions are transposed to the cluster level a second force component (at the direction of the cluster) is temporarily added to the destination pulling force, influencing the resulting motion of the packet along the norm of the net addition of the two components. As this happens implicitly and still under a greedy regime, it is significantly less complex and problematic than face or other hybrid routing methods for recovering from local minima[1]. This latter effect, which is a direct consequence of the first, has also been pointed in [39].

The confirmation of the positive impact that these effects have on greedy routing has inspired some ideas for follow up work. The first relates to resolving the problem of routing loops in a regime that would allow continuous hopping between embedding levels (see section 3.4.4). This practically would make the benefits from the first effect more pro-

[1] Although to conclude the absolute superiority we would have to design a suitable routing performance benchmark for this comparison

nounced. One possibility towards this direction may be to consider an in-packet path history mechanism similar to the one used in [41].

One additional exploration would be to compare the performances of our approach with combinations of different embeddings of the network graph that are not necessarily cluster views of it. One could imagine routing on two different embeddings of the base level graph, referring to one if a dead-end is reached on the other. This would be somehow similar to the notion of different *realities* in [90]. In the same way one could picture a comparison with a three-level routing in which the second and third levels are both embeddings of a cluster view of the network graph using different clustering methods.

3.6.2 Clustering versus inter-domain routing

Drawing on the similarity of the two level identifiers (cluster level, node level) with network and host identifiers in the Internet, one may be tempted to think that there is a similarity between the work we present here and hierarchical routing techniques [65] or inter-domain routing [17] in the Internet. However, this similarity is neither conceptual, nor functional. Network hierarchies in classical and inter-domain routing serve as a measure to aggregate and reduce the amount of control information required for the operation of stateful routing. No such need exists in geometric routing.

Also, a major advantage of our method over classical hierarchical addressing is that we do not hide away proximity information at the low network level. Consider the example of figure 3.14. In figure 3.14a, a classical hierarchical addressing scheme was used in which a node's full address consists of the concatenation of the address of the cluster to which the node belongs with the node's address within the cluster. A particular disadvantage of such an addressing scheme is that the obtained addresses can only indicate proximity within the boundaries of the cluster. When provided two nodes' addresses 1.1 and 1.3 one can deduce that the two nodes are close to each other as they belong to the same cluster. However, and as shown by the example two nodes in figure 3.14a, two direct neighbours may have very distant addresses if they happen to belong to different clusters. Such a situation is well known in classical networking and is accounted for by the use of routing state. In contrast, in our hierarchical embedding addresses, the proximity of nodes' addresses is maintained beyond the cluster borders. Consider the example addressing of figure 3.14b. In this example, the network

connectivity graph is first embedded (without any clustering) and both nodes A and B are attributed the two-dimensional coordinates $(3, 2)$ and $(3, 3)$ respectively indicating a proximity between the two nodes. At a second stage, the cluster graph is in turn embedded in a two-dimensional space, resulting in Cluster 1 being attributed address $(1, 0)$ and Cluster 2 $(2, 0)$. Since, the full address of a node in our model is the concatenation of its coordinates at the different levels of the graph, node A's full coordinates then become $(1, 0), (3, 2)$ as shown in the figure. When comparing the addresses of the two nodes A and B in our scheme, one has simply to ignore the cluster-level part of the addresses in order to have an estimation of the distance between the two nodes at the flat network level.

Functionally, another major difference is that our proposal considers routing in both directions of the hierarchy (from cluster level to node level, as well as from node level to cluster level). In classical and inter-domain routing approaches, routing always starts at the aggregate level, and concludes at the host level.

3.7 Conclusion

In this chapter we have provided some insights on the effects of clustering for improving the performance of greedy routing. We looked at clustering from a general perspective (as opposed to focusing on a specific method), and we proposed a framework that enables cluster based greedy-routing over any low-dimensional embedding. In our evaluation, with clustering methods of varying sophistication, we measured and reported significant improvement of up to 25% (1-level clustering) and 35% (2-level clustering), over a number of reference systems in the literature. We experimented with a variety of configurations (different network sizes and densities), so as to develop a better understanding on the effects of clustering, and quantify its overheads over the stateless nature of geometric routing. Finally, based on the insights we acquired, we have identified a number of interesting directions for further exploration and follow work.

Chapter 4

Greedy Routing and administrative policies

4.1 Introduction

Greedy geometric routing strategies were initiated and became popular within the ad'hoc and sensor networking communities [63],[13],[89]. Not only these techniques seem to be more natural in the context of local wireless communication, they also offer the advantage of considerably reducing the amount of control information storage required for the functioning of the routing process. This routing table compression capability constitutes a major favorable argument in the case of sensor network nodes where memory usage is of critical importance. Greedy routing techniques achieve this goal by introducing the notion of proximity in the network addresses. Indeed, in all greedy routing proposals, the network addresses are attributed to the network nodes so as to reflect their vicinity; i.e. two network nodes that are only a few hops away from each other, will be attributed two network addresses that are considered close to each other given a chosen distance function.

Since their introduction, greedy routing methods have considerably evolved to support many interesting features. Properties such as guaranteed message delivery ([64], [40]), scalability to a large number of nodes [41],[20] and autonomous address attribution (and re-attribution in case of changes) ([89], [75]) are challenges that have been successfully addressed by the research community.

Given the impressive list of advantages mentioned above, geometric greedy routing techniques started lately to gain attention outside of the usual communities where they hatched out. Indeed given the constant rise in the amount of control information that current internet routers have to store in their routing and forwarding tables (more than 300 000 forwarding entries per-router for IPv4 [1]), the idea of shrunk routing tables sounds very appealing. Therefore, greedy routing is more and more considered notably within clean-slate efforts as a possible replacement for the current standard routing techniques [71].

Our belief is however that, in their current state of the art, greedy routing techniques cannot be directly applied for large scale networks similar to the current Internet. Note that the problem there is by far not scalability. Indeed this issue has been identified and targeted since very early works in the field [19, 41] and some greedy routing techniques show very good performance with a large number of nodes. In our view, the problem comes from the fact that greedy routing mechanisms have been mainly designed and used for uni-corporate networks such as a sensor network of a single University or company or a collaborative ad'hoc

community. In such use-cases, all the nodes forming the network belong to the same administrative entity and therefore share the same goals. In such circumstances, greedy routing methods have not so far been challenged by any administrative incentives issues where multiple parties, with conflicting priorities, are involved in constructing the network.

Note that this will most certainly not be the case when greedy routing is applied to a network similar to the current Internet. Indeed in such a network and as demonstrated by the BGP specification, special measures have to be considered in order to satisfy the constraints of different parties. A pure and simple routing method based solely on the proximity of the machines might be scalable, but will most certainly not be sufficient when communications are subject to security, quality of service, customer-provider and pricing concerns. Our goal in this work can therefore be pictured as trying to set up the basis for a Border Gateway Protocol specific to geometric greedy routing techniques so as to get these techniques one step closer to a large scale deployment.

We start our study by a definition and brief state of the art of networking policies and routing with administrative constraints. We then introduce via an example our new administrative challenge that we believe to be a requirement for the future development of these techniques. Finally we propose a set of possible solutions that will be detailed in the following chapters.

4.2 Administrative relationships and policies

Let us start by first clearly defining the wide notion of networking policies and specifying the type of policies that we will be focusing on.

Policies in communication networks are the mean by which network administrators (or users in general) indicate their preferences and requirements to the software agents performing the communication tasks. Due to the fact that a user's preference panorama can be rather large, consequently, networking policies can be quite diverse including (but not limited to) security policies, quality of service or traffic policies and administrative policies.

4.2.1 Security policies

In this case the user or the administrator uses security policies to express preferences regarding the right of access and modification that is attributed to users or software agents with regard to services and data. A typical example would be to limit the access to a critical service to a particular set of users (or agents). In current Internet solutions, this is done by instructing the host on which the service is residing to disregard any connection attempt initiated by machines other than the listed authorized ones (using software firewalls for example). In the more specific context of routing, security policies can be used to avoid "dangerous areas" of the network. Indeed, in the case of possible multiple paths to a destination, policies can help choose the safest one avoiding to traverse nodes known for spying on transiting data. The support of such policies requires a visibility of the full packet path which in turn requires a functioning routing mechanism. Hence we consider the support of such policies as future work.

4.2.2 Traffic policies

Policies can also be used in the context of quality of service and traffic provisioning. In this context, administrators can specify quotas attributed to users (or network neighbours) regarding bandwidth or total amount of data traffic. Such policies are usually implemented "on top" of routing, meaning that the policy verification mechanism operates as an add-on to the routing process and is not inherent to it. Therefore also in this case, the priority is given to the achievement of a functioning routing mechanism before focusing on the traffic issues.

4.2.3 Administrative relationships and policies

Administrative policies are our main concern in this chapter. Administrative policies play a role in networks subject to financial agreements between different providers.

Administrative relationships A typical example of a network subject to administrative relationships is the Internet. In this case the global network is formed by the concatenation of smaller networks operated by different Internet Service Providers. These providers being for the most private, profit-oriented companies, packet transits between the different constituent networks are subject to monetary issues. Hence, a

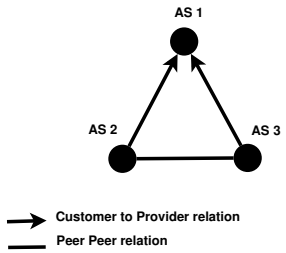


Figure 4.1: Relationship Between Network Providers

link between two networks belonging to two different service providers can usually be attributed one of the following relationships defined by [47] : customer-provider, peer-peer and sibling-sibling relationship.

In the customer-provider relationship, one of the two service providers is purchasing network access from the other. The network purchasing the connection is referred to as the *customer* while the other is known as the *provider*. In such a relationship the customer network pays the provider for both incoming and outgoing traffic.

In the peer-peer relationship, the two network providers agree to establish a connection that is not subject to financial tariffing. Such an agreement usually happens when the two networks transfer through each other approximately the same amount of traffic in both directions. A Peering link might in some cases be specifically created between two frequently communicating networks to avoid them both having to pay providers in order communicate with each other.

Siblings relationships are usually the result of a merger between two Internet Service Providers or can also be the result of traffic engineering. A sibling relationship between two independent networks usually means that they in fact belong to the same authority that operates both of them. For the rest of the chapter we will ignore sibling relationships as they are a relatively rare phenomenon and since the two (or more) sibling nodes can simply be represented by a single node (an aggregation of the two sibling nodes) with as neighbouring nodes the union of the neighbours of the component nodes.

Figure 4.1 depicts a graphical model for representing administrative relationships between different providers as introduced by Dimitropoulos et al [35] . In this figure, a directed edge indicates a customer-provider relationship between the 2 nodes, initiating from the customer and pointing towards the provider. The direction of the edge hence

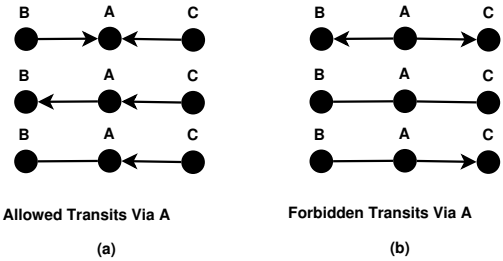


Figure 4.2: Allowed and Forbidden Crossings that A offers to neighbours

indicates the *cash flow* from the customer towards the provider. The peering relationships between two nodes is depicted by an undirected edge indicating that there is no monetary exchange between the two. Hence, in this figure, *AS 2* and *AS 3* both purchase connectivity from the provider *AS 1*. *AS 2* and *AS 3* also maintain a peering relationship with each other.

Note that this is simply a logical representation of the relationship between the different network players and does not represent data flow direction restrictions between the different nodes. All that can be inferred from an edge in a diagram such as in figure 4.1 is the presence of at least one network link between the two nodes allowing data to flow in both directions.

Administrative policies *Administrative policies* are policies defining routing decisions in a context subject to administrative relations between the network operators. Indeed, in such a context, policies are for example used to discriminate between multiple possible paths to a destination based on the administrators preferences. In BGP this is done by assigning a preference value to a provider that is usually correlated with its monetary cost. This way, when multiple paths to a destination are possible, the routing agent can favour the cheapest providers leaving the more expensive ones for backup uses only. In the same logic, paths to a destination that are transiting via peers are preferred to paths via providers as they don't incur any communication costs. Similarly, paths transiting via customer nodes are usually the most preferred as they do not imply monetary costs and they do not burden the peering links. Indeed, relying too much on a peer for transit might create an imbalance in the traffic load between the two, pushing

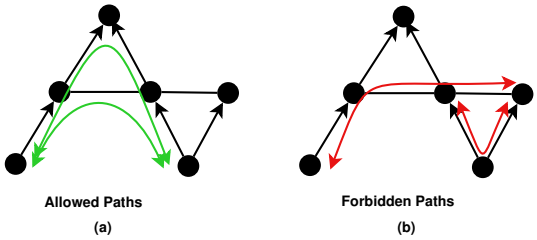


Figure 4.3: Valley-free paths

the peer to re-examine the relationship and possibly terminate it.

Administrative policies also determine how routes are advertised in the network which in turn determines how packets are forwarded. Based on the relation with two of its neighbours, a node might decide to offer or refuse to transit packets between the two. A typical example is a customer operator refusing to transit communications between two of its providers as this would induce costs in both directions, all for a communication in which the operator has no interest. Figure 4.2 shows a list of the most commonly used transit rules in the Internet that are deducible from economic incentives. In the list of forbidden transits in Figure 4.2.a, one can see that node A would refuse to transit packets between two of its peers. This might sound strange as one would think of a peering relation as an agreement to share all links and hence offer all possible transits. However, in a peering relationship as it is used in the current Internet, a node offers a transit towards its peer only to its customer nodes. Any other type of (non-paid) communication is considered a burden on the peering link and is therefore denied.

Note that the transit authorization works for communications in both directions. Hence in the authorized cases of figure 4.2.a, data can traverse node A in both directions, namely $B \rightarrow A \rightarrow C$ and $C \rightarrow A \rightarrow B$.

Given that a path between two nodes in a network subject to administrative constraints would be a concatenation of such transits, routing in a network similar to the internet exhibits a pattern known as "Valley-free routing". This pattern suggests that any route between two nodes is formed by zero or more customer-to-provider links, followed by zero or one peer-peer links and finally followed by zero or more provider-to-customer links. In a representation such as Figure 4.3, in which provider nodes are drawn above customers and peer nodes are drawn

at the same level, the path taken by a message would first travel upwards following customer-to-provider links, then horizontally according to a peering link, and finally downwards following provider-to-customer links. All paths in which the message would be moving downwards first and then upwards again are forbidden according to the transit rules of Figure 4.2, hence the name *valley-free*. Note that due to the fact that *peer* $\rightarrow$ *peer* $\rightarrow$ *peer* paths similar to that in figure 4.3.b are also forbidden, a denomination as *valley and plateau free routing* would be more appropriate.

4.3 Administrative policies and future networks

Very large networks, the size of the Internet cannot possibly be created and managed by a single organization. Indeed such large networks are usually the result of a long-lasting constructive effort involving multiple parties.

A viable example to this statement is the current Internet or more precisely the inter-domain network. As mentioned above, the structure of the Internet can be divided into several interconnected but independent large networks that are the Autonomous Systems, mainly managed by large companies (*AT&T, DuPont*, ..) and Universities.

Due to the fact that the deployment and maintenance of such a large network infrastructure induces considerable costs, the interconnection and packet transiting between the Autonomous systems is consequently subject to business agreements. This fact is most likely to remain true for any upcoming large-scale network to be developed, be it an extension of the current Internet, or a fully new global network. Therefore, the above stated relationship agreements that are the customer-provider and peer-peer relationships are not likely to faint away and will most probably affect the way data is transferred in future large networks.

4.3.1 Administrative policies and greedy routing

Sensor networks as well as ad-hoc networks communities in which greedy routing developed, have so far mainly considered simple environments where the network belongs entirely to a single entity, be it a company or a University. In such an environment, all the nodes taking part in

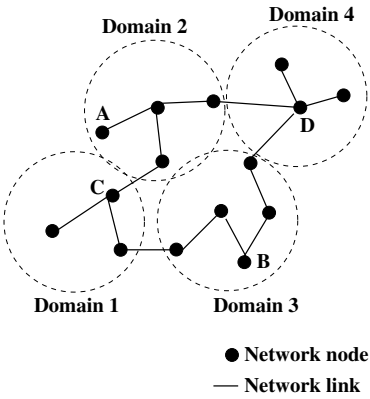


Figure 4.4: Multi-party Network

the network activity share the same goals and will generally collaborate towards the good functioning of the network.

The context that we consider in our case is a more challenging one where the network is not formed by a single party but is instead built by several network providers agreeing to collaborate to build a larger network. In an ideal world, these providers could agree to equally share their resources allowing packets to flow freely between the different network domains. In such a case the network could still be considered as a single entity network belonging to the "Confederation" of providers, and the current state of the art greedy routing strategies could be directly applicable to it.

Sadly, and as argued above, it is very unlikely that the different parties building the network will offer free full access to their resources. Instead, it is most probable that these parties will develop complex interactions pretty much similar to those found in BGP/AS interactions and will be trying to protect their interests in a networking context subject to security, traffic engineering and pricing issues.

To illustrate this, let us consider the example in Figure 4.4. In this case the communication network is built by four different providers each of which is responsible for the nodes within its domain as indicated in Figure 4.4. What happens if the domain providers do not intend to fully collaborate ? What if there are security or pricing concerns ?

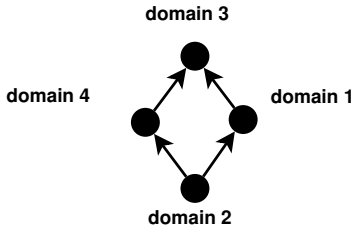


Figure 4.5: Administrative relations between domains of Figure 4.4

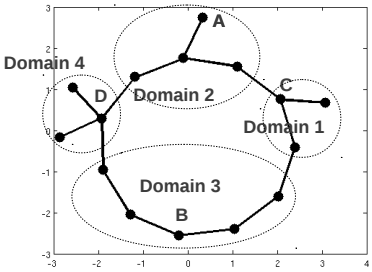


Figure 4.6: MDS embedding of network in Figure 4.4

Let us suppose that the provider of *Domain 4* decides to economically charge the nodes of *Domain 2* for any transiting packets (i.e. packets not destined to *Domain 4* nodes). According to the terminology defined above, in such a setting *Domain 4* can be referred to as *a service provider* to *Domain 2*. Reciprocally, *Domain 2* is referred to as the *customer* of *Domain 4*. Let us also suppose that *Domain 1* is a provider for *Domain 2* with cheaper transit fees than 4. Finally, both domains 1 and 4 are customers of domain 3. Figure 4.5 summarizes the administrative relationships between the domains of Figure 4.4.

In such an environment, imagine now if we were to attribute addresses to the network nodes based on their non-administrative network distances with the intention of performing greedy routing. Figure 4.6 shows such a possible address attribution computed using metric multidimensional scaling [12] method in a two-dimensional Euclidean space.

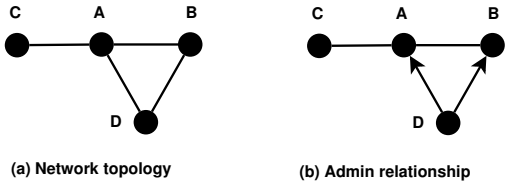


Figure 4.7: Example topology and its administrative relationships

In such a context, If we are now to route, for example from a node A residing in *Domain 2* to a node B residing in *Domain 3*, greedy routing will be transiting through *Domain 4* that has a shorter hop count to *Domain 3* than the route via *Domain 1*. Considering that *Domain 1* offers a cheaper transit to *Domain 3*, relying only on greedy routing is clearly not in the interest of the provider of *Domain 2*. We refer to this issue as *the preference support problem*.

A more important problem appears in our setting when routing from node C in *Domain 1* towards node D in *Domain 4*. In such a case, the greedy routing procedure will direct the packets towards the route via *Domain 2* as it would be the one with the shortest path. However, since in the current setting *Domain 2* is a client of both 1 and 4, it will most certainly not offer them a transiting service for which it would have to pay. Therefore, the packets originated at *Domain 1* destined to *Domain 4* will be bounced back when entering *Domain 2* creating a constant point of failure at node *C* in Figure 4.4. We refer to this issue as *the policy dead-end problem*.

4.4 The problem from a graph viewpoint

The introduction of administrative relationships in the network creates problems as it implies that the different operators all have a different view of the network graph. In order to illustrate this, let us consider the example network of Figure 4.7.a. The vertices in this figure represent Internet Service Providers. Figure 4.7.b represents the administrative relationship between the different providers.

According to the transit rules depicted in Figure 4.2 let us consider the possible paths that packets emanating from node *B* can traverse. First of all one can notice that packets emanating from *B* and destined to *C* will be refused transit on both nodes *A* and node *D* since *A*

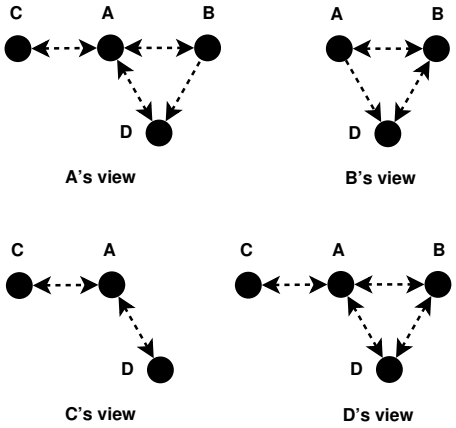


Figure 4.8: Different views of nodes on the same network

has a peering relationship with both B and C, and D's only way to C is via A, thus implying a provider to provider transit. Hence in B's view of the network, node C is disconnected or might as well be invisible. Also, although a connection loop exists between the three nodes A, B and D, packets starting from B cannot traverse the loop in all directions. For example, B can reach node D via the path $B \rightarrow A \rightarrow D$ (as D is a customer of A). However, the path $B \rightarrow D \rightarrow A$ is not possible as it violates the *valley-free* rules of administrative routing. Hence this induces directionality in the edges seen by a particular node in the network. The representation of a node's view of the network can therefore be modelled using a directed graph in which an undirected edge of Figure 4.7.a would be represented by two directed edges pointing in opposite directions.

Figure 4.8 shows *B's* view of the network along with the view of the other nodes. One should notice the striking difference between the different views in which vertices and edges can be missing. For example, one can see that also in the case of node C, only a partial view of the network is obtained as packets originating from C cannot reach B and vice versa. In the case of node A, only an edge from D to B is missing as the path $A \rightarrow D \rightarrow B$ is blocked by D. The only node with a complete view of the network graph, i.e. the packets of which can freely traverse any path of the network in this case is node D.

Representing the effects of administrative relationships on the per-

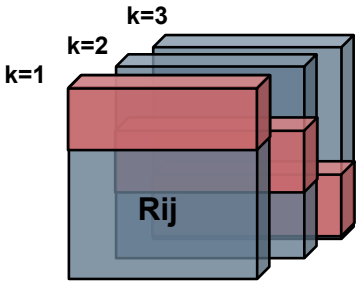


Figure 4.9: Example of a row-dominated LAS on a 3x3x3 incidence matrix. The red rows are selected as consensus values.

ceived network topology in such a way helps getting an intuition about the complications that it adds to greedy routing. In order to allow for greedy routing, an embedding of the network graph has first to be realized in a given metric space. All nodes participating in the greedy routing process, will then base their routing decisions by computing distances between the coordinates of the same embedding. In a network non-subjected to administrative relationships, the network graph is a single undirected graph upon which all nodes agree. Hence, provided that the embedding correctly matches the network graph, any routing decision made by a node in the embedded world will automatically match its view of the network.

The complication in our case, and as demonstrated by Figure 4.8, is that each node participating in the greedy routing process has a different perception of the network graph. And yet, the classical approach of greedy geometric embedding requires that there should be a single embedded world (coordinate-space) upon which nodes should base their greedy routing decision.

The problem is then of finding such an embedding that satisfies all the different views at the same time. Such a problem is in fact analogous to embedding multiple graphs into a single metric space.

4.4.1 Relation to social sciences and Cognitive Social Structures

A similar phenomenon involving multiple graph views was previously studied in the domain of social network analysis. In a 1987 paper [69], Krackhardt proposed the notion of *Cognitive Social Structures (CSS)* that arise in a context where multiple individuals taking part in a social network are related by an asymmetric relationship such as "A considers B as a friend" or "A asks B for advice". Note that these relationships are by nature asymmetric since they represent social connections that are not necessarily reciprocal since having a person A considering another person B as a friend does not necessarily mean that person B feels the same about A. Given this setting, and assuming a complex network in which an individual cannot possibly keep track of all the existing relationships between the other network participants, and adding the fact that each individual can differently perceive the manifestation of a relationship between two individuals, each participant in the social network will develop a different view, or perception, of the global network.

In an empirical experiment where different managers belonging to the same company were asked to report on "who in their view asks for advice from whom ?" (also an asymmetric relation in this case), each individual gave an impression of the network that is different from that reported by other participants to the experiment. The reported networks consisted in multiple directed graphs, similar to those arising in the case of administrative routing, denoted as *cognitive social structures*.

Social scientists were interested in CSS in two manners. The first study approach is to consider the different views as a phenomena to study independently. One can for example study the centrality of the perceiver node in his perceived network and compare it to the centrality value for the same node as reported in other participants' view. Such a comparison would allow to determine the ego bias in an individual's opinion about his role in a community. Another approach to study the CSS structures is that of finding a consensus structure between the different views. The motivation for such a consensus can vary from uncovering the ground truth in case this couldn't be measured, to building a more tractable data structure allowing to study the social network more easily.

Two main approaches for finding a consensus structure exist. The first approach named *Locally Aggregated Structures (LAS)* builds a common network by selecting component parts of the different network

views. More specifically, the chosen common relation in the consensus graph between two nodes i and j is mainly depending on the views of the concerned nodes i and j (hence the *local* denomination). Following the notation of Krackhardt, in a network defined by an asymmetric relation R where $R_{i,j,k}$ when set to one indicates the presence of a relationship from node i to j in the network perceived by node k ($R_{i,j,k}$ set to zero otherwise), and denoting by R' the relation defining the consensus graph, the LAS approach can be divided into :

$$Row\ Dominated\ LAS:\ R'_{i,j} = R_{i,j,i}\ ,\ and$$

$$Column\ Dominated\ LAS:\ R'_{i,j} = R_{i,j,j}$$

When representing the different network views as a three-dimensional matrix in which the two first dimensions describe a typical incidence matrix and the third dimension indicates the perceiver node, both the row and column dominated LAS can be pictured as a diagonal slicing of the three-dimensional incidence matrix as can be seen in figure 4.9. Another form of Locally Aggregated Structure is to consider boolean functions between the values of the concerned nodes i and j such as :

$$LAS\ from\ intersection\ rule:\ R'_{i,j} = R_{i,j,i} \cap R_{i,j,j},\ and$$

$$LAS\ from\ union\ rule:\ R'_{i,j} = R_{i,j,i} \cup R_{i,j,j},\ and$$

where the intersection rule dictates that a relation between i and j is to be reported in the consensus graph only if it is present in both views belonging to i and j. In the case of the union rule, the relations needs to be reported in the view of only one of the two concerned nodes in order to be included in the consensus graph.

Another approach to obtaining a consensus structure is to consider the relation between i and j as a function of all the available perceptions of this relation. Namely, $R'_{i,j} = f(R_{i,j,1}, R_{i,j,2}, ..., R_{i,j,n})$ where n is the number of participants in the network. A typical such function is the majority vote according to which a relation is represented in the consensus graph if more than half of the participants do perceive it.

4.5 Understanding the problem from a distance matrix viewpoint

$$A's\ view: \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 2 \\ 1 & 2 & 2 & 0 \end{pmatrix} \end{array} \quad B's\ view: \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 1 & \infty & 1 \\ 1 & 0 & \infty & 1 \\ \infty & \infty & 0 & \infty \\ 2 & 1 & \infty & 0 \end{pmatrix} \end{array}$$

$$C's\ view: \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & \infty & 1 & 1 \\ \infty & 0 & \infty & \infty \\ 1 & \infty & 0 & 2 \\ 1 & \infty & 2 & 0 \end{pmatrix} \end{array} \quad D's\ view: \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 2 \\ 1 & 1 & 2 & 0 \end{pmatrix} \end{array} \tag{4.1}$$

One could also look at the problems incurred by administrative relationships from a point of view complementary to the graph one that is that of distance matrices.

Given that each node participating to a network subject to administrative relationships will have an independent view of the network (graph), this also implies that it will have a distance matrix describing the distances between the nodes of that graph. The challenge of finding an embedding to allow for greedy routing on such a network becomes the one of finding an embedding in which the distances between the embedded nodes match those belonging to multiple distance matrices. Table 4.1, shows the different distance matrices that correspond to the views of the nodes participating in the network of Figure 4.7.

A first noticeable challenge to the task of finding an embedding are the asymmetries that can be seen in the matrices corresponding to a given node's view. For example, one can see that the distance from node D to A in node B's view is different from the opposite distance. This can be explained by the fact that node D refuses to offer transit towards A to B and hence in A's view, D's distance to B is equal to A's distance to B incremented by one hop that is 2. However, given that node A offers transit towards D, it announces to B its real shortest distance that is 1.

Note that these asymmetries are the logical consequence of the directed edges that can be seen in the partial graph views of each node, as packets initiated at the node to whom the view belongs can travel along some edges in one direction and not the other.

Another difficulty visible in Table 4.1 are the incoherences that appear between the different matrices. Consider for example the distance from node D to A in the two matrices belonging to B and C's views. In this case the two nodes A and D should appear close in the view of node C and distant in the case of B's view. This clearly complicates the task of finding an embedding in which to perform greedy routing, as it is difficult to find two points in a metric space that are at the same time close and far from each other.

Another point to be made about the different "matrix views", that might ease the process of finding an embedding, is that the nodes participating in such a network do not require to have a full distance matrix detailing the distances from every node to every other. In order to reach a decision on where to forward a packet to a given destination, a node is required to know only the distance from its neighbour towards the destination, in order for it to select the neighbour closest to destination. Hence only a partial view of the distance matrix is required, namely the rows of the distance matrix belonging to the neighbours. This removes a big part of the difficulties related to the asymmetries in the distance matrix. The problem of finding a greedy embedding that satisfies administrative relationships becomes then that of fitting multiple partial incoherent distance matrices into a single metric space.

4.5.1 Relation to tensor decomposition and 3-way multidimensional scaling

When considering the multiple views problem from the distance matrices angle, the problem can be related to that of tensor data decomposition and more precisely 3-way multidimensional scaling [12]. Tensor data analysis is concerned with data in the form of multidimensional arrays where a vector is considered as a one-dimensional or one-mode array, a matrix as a two-mode one etc. Such datasets are frequently encountered in fields such as psychometrics, linguistics and chemometrics. The most common approaches for studying such a data structure rely on generalizations of the matrix (two-mode tensors) operations such as Singular Value Decomposition and Principal Component Analysis to n-mode data structures such as the *Canonical Polyadic Decomposi-*

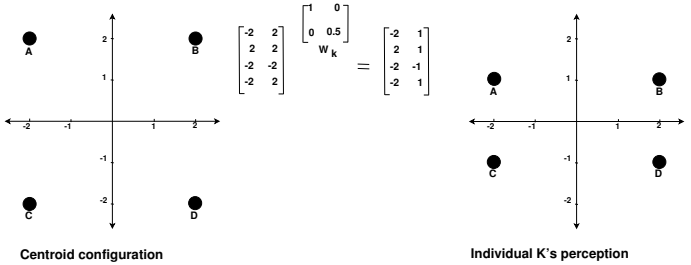


Figure 4.10: Example of Individual differences model through dimension weighing

tion (also known as *PARAFAC* and *CANDECOMP*) and the *Tucker3 method* [67]. All these methods share the common goal of obtaining transformations on the data allowing for a compacter representation, in a similar way PCA uncovers data features in order to represent it in a lower-dimensional space and thus obtain a lossless compression. Lossy compression would also be possible when using the above mentioned methods as they allow for a spectral representation of the tensor data and thus the possibility of incrementally deleting irrelevant details.

Such methods are generally applied to generic data types or *feature vector data*, where the data represents characteristics of the described objects, or, differently said, when the objects generating the data are already embedded in a high-dimensional space. When the data however describes similarities or dissimilarities between a set of objects (the spatial configuration of which is unknown), it is better analysed in the framework of *3-way multidimensional scaling*. In this particular case of MDS, the provided data to be embedded in a metric space consists in a three-dimensional data structure where the third dimension defines the *perception* of different individuals of the distances. Such datasets arise frequently in the field of psychometrics where one measures the reactions of several test subjects to different stimuli. A typical example of such a test is the case of Helm's colour data [57]. In this test, 14 individuals (test subjects) were questioned about the similarity of 10 different colours ranging from red to purple. An intuitive data collection scheme was then used in which the participants assigned the similarity values *physically* by placing a coloured chip on a surface with regard to two other differently coloured chips so as to reflect the colour similarity. Due to the specifics of the data collection device and to the presence of

both colour-normal and colour-deficient subjects, such a test results in a three-dimensional data array where a value $D_{i,j,k}$ indicates the distance (difference) between colours i and j as perceived by participant k. In the MDS literature, several approaches exist for embedding such a data structure. A first method denoted as *Generalized Procrustean Analysis* consists in first performing separate embeddings of each of the participant's perceptions of the distances and to later-on conduct procrustean analysis between all pairs of embeddings. In a procrust operation, the goal is to *fit* a given point configuration (known as the *testee*) into another configuration (the *target*) through the use of both rigid motions (i.e. transformations maintaining an object's shape such as translations, rotations, reflections) as well as dilations and linear distortions. As a result, a "centroid configuration" can be computed such that it minimizes the differences to all of the transformed embeddings. The advantage of such an approach is that each node's perceived embedding can then be approximated by performing a transformation of the agreed-upon centroid configuration. This allows us to encode the *individual differences* of each participant in a single transformation. The major drawback of the method however is its computational complexity. Indeed, given a set of N perceivers, this method requires the computation of N separate embeddings and performing an optimization operation simultaneously minimizing the errors of $N(N-1)$ procrust operations. Therefore other methods such as *INDSCAL* [12], have been developed to search directly for a centroid configuration as well as the transformations for each of the different perceivers. However in this case, the transformations are limited to linear distortions of the centroid configuration or in other words applying weights to dimensions when evaluating the distances within each node's perception. Figure 4.10 shows an example of the dimension weighting approach. In order to evaluate the distance between two points in a given participant's perception, a linear distortion (dimension weighting) of the centroid configuration is performed by multiplying the matrix corresponding to the agreed-upon configuration with a diagonal matrix W_k corresponding to the dimension weights of the perceiver. The perceiver's distance between two nodes can then be evaluated in the transform result. Other methods such as *IDIOSCAL* allow for non-diagonal transformation matrices therefore making combined operations such as translations and rotations also possible.

A possible angle of view on the above mentioned operations can be taken from the distance function perspective. For example, the *INDSCAL* operation results in that different weights are applied to the point

coordinates' dimensions when computing the Euclidean distance. In other words, this method allows to find, for each node, a different distance function that is more appropriate to its perception of the dissimilarities. In chapter 5, we explore the feasibility of applying a different distance function per network node, without however restricting our search for the distance function to a weighted Euclidean one.

Another possibility of dealing with tensor data for 3-way MDS applications can be to *unfold* the multidimensional data structure. This operation transforms the N-way data structure (in our case 3-way) into a more common 2-way data structure : a matrix. Common multidimensional scaling tools can them be applied to find an embedding corresponding to the obtained matrix. In chapter 6, we discuss the use of possible existing such transformations and we introduce our own transformation that is more suited for our greedy routing needs.

4.6 Why is there a requirement for a single embedding ?

Given that each node's perception of the network graph can itself be interpreted as an independent graph of its own, a legitimate question would then be to ask why not embed all these graphs separately and then perform greedy routing upon them ?

The problem with this approach is that of mapping between addresses belonging to the different embeddings. Assume that nodes A, B and C of the example figure 4.7, each embed their views of the network separately. Suppose now that node C wishes to send a packet to node D. This requires that node C knows the address of node D in its own embedding and that neighbour A's address is closer to D's coordinates. When the packet is then forwarded to A, the destination address of the packet is that of node D in C's embedding. However, in order to perform its routing decision, node A requires to know node D's address in its own embedding. This address will most probably be different from the one attributed to node D in C's embedding.

Hence two options are possible. Either have a coordination mechanism between all the nodes so that the coordinates attributed to all the nodes in the different embeddings are the same, i.e. node D has the same coordinates in the embeddings of both C and A. In this case, we are back to the single agreed upon embedding fitting all the different views. Or, we need to implement a mapping system between the different "real-

ities", allowing to translate a node's address within a given embedding into its address within another embedding. Such a mechanism could in principle be implemented through the use of unique identifiers upon which all nodes agree. Messages would then be sent to nodes using their unique identifiers. Each node relaying the packet would then map the destination's identifier to an address in its own embedding in order to perform a greedy routing decision. Such an identifier/address mapping system would then require each node to store both an identifier and an address for every other node in the network. Note that this defies the purpose of greedy routing in the first place, as the main goal of using greedy routing was specifically to limit the size of routing tables by avoiding to store an entry for every existing node. Even more, if we are to store such an id/address table, one can for the same storage cost replace it by a id/next hop table, hence avoiding the risk of failure incurred by greedy routing.

4.7 Strategies for solving the preferences problem

As mentioned previously one of the two major failures of greedy routing in a network subject to administrative constraints is its failure to support node preferences in routing. Indeed as demonstrated by the example of figure 4.4 in some cases the shortest path to a destination is not necessarily the most preferred one, especially when monetary constraints define the way routing is to be done.

Two approaches to solve this problem can be taken. The first approach consists in finding an embedding that encodes the preferences of the nodes in the distances between their attributed coordinates. Expressed differently, in case two next hops are possible towards a given destination, then the preferred next hop should appear closer in the embedding to the destination than the other, in order to be selected by the greedy algorithm. Such an approach however complicates the embedding procedure as it adds additional constraints on the coordinates to be attributed to the network nodes. The methods for finding an embedding that satisfies such constraints on the node coordinates will be discussed in chapter 6.

A second approach to tackle this issue is to store the node preferences locally in a similar way *BGP* does. Whenever a node is confronted with a choice between two possible paths, it can then simply consult its

locally stored preference list, and forward the packet to the indicated preferred next hop. The advantage of this method being that the encoding of the preferences and their usage is done locally at the forwarding node, and does not require a cooperation between the different nodes in order to find coordinates matching the preferences of all parties. A necessary condition however to the feasibility of such a solution is that the node is able to determine the possibility of multiple paths to a destination. Determining such a possibility in a greedy routing environment is equivalent to having both possible next hops appearing closer to the destination node in the embedding than the current forwarding node is. Finding an embedding that satisfies such a property can reveal to be a quite difficult task as shown in chapter 6. Indeed, the classical approach of performing such an embedding, namely multi-dimensional scaling, relies solely on the shortest path distance matrix of the graph to assign the nodes coordinates. However, relying only on the shortest distance for the embedding can result in hiding the presence of other possible paths.

4.8 Strategies for solving the policy dead-end problem

The main problem of greedy routing in a network subject to administrative relationships is that of being lured towards forbidden paths. As argued above, such a problem is the result of incoherences or disagreement between the network nodes. An edge or a distance, depending on the way we are looking at the problem, might appear differently to different nodes. Consequently two approaches could be taken to solve this.

A first approach, inspired from the cognitive social structures ideas would be to find a consensus situation on which all the nodes agree. This consensus can satisfy the requirements of the nodes either fully or to the best feasible. An example of such a consensus is to have all nodes agree on using the *same* distance matrix for the embedding. Approaches for finding a single distance matrix satisfying the requirements of all nodes are studied in chapter 6.

A second approach to solve this issue is that of replication. Instead of finding a consensus, the disagreements between the different nodes can be solved by replicating nodes and edges. In case nodes disagree on the directionality of a given edge, or on the distance between two nodes,

the problem can be solved introducing replicas of the edges or vertices, each satisfying the requirements of a given node. This approach divides in two subgroups in which the replications are done either before or after the embedding. In the first case, the network graph is modified in order to better exhibit the different views implicated by the administrative rules. The hope is then that an embedding of the modified graph containing replicas would reflect the administrative constraints. In the second case, replicas are introduced in the embedding space through the use of virtual points of presence where a node announces different geometric positions (coordinates) to his neighbours depending on its interest of being perceived closer to a given set of destinations or not.

A third way of approaching the problem is to consider the possibility of breaking up with the idea of a geometric embedding space and that of having all nodes share a common distance function. This method, that is also partly inspired from the 3-way MDS methods, and most precisely the dimension weighting techniques, is studied in chapter 5, where after agreeing on a set of global identifiers for the nodes, each network node *searches* a distance function that is describing at best the distance information communicated by its neighbours.

4.9 Roadmap

In the coming two chapters, we will be addressing the policy dead-end problem from two different angles apparented to those identified above. An important remark however, is that for the rest of this thesis we will be focusing on the problem of greedy routing at the domain-level. This means that all of the network participants that we will from now-on refer to as *nodes* are in fact network domains representing all underlying nodes similar to those of figure 4.5.

Our choice of doing so is motivated by the above described complexity of the problem of greedy routing on administrative constraints and the abundance of branches to explore. Thus it is more convenient for now to explore the effects of each possible solution at the domain-level graph first. Note that this does not however hinder the possibility of an administrative greedy routing solution as a two-levelled greedy routing procedure similar to that proposed in chapter 3 could be possible. In such a scenario, one of our proposed solutions for greedy routing at the inter-domain level could be coupled with the classical geometric greedy routing at the intra-domain level. In a similar way to the exam-

ple figure 3.5 in chapter 3, intra-domain routers would be required to maintain two distinct neighbourhood tables for both the intra-domain and the inter-domain graph (analogous to the cluster-level graph).

Chapter 5

Geometric areas for policy support

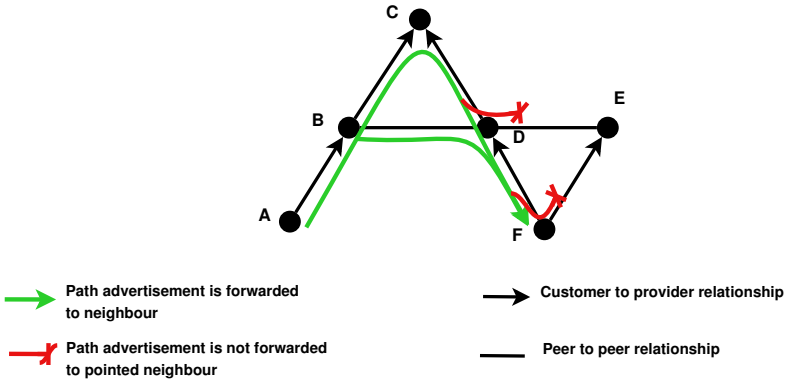


Figure 5.1: Propagation of NLRI announcement initiated by node A

In the previous chapter, we identified two main problems preventing greedy geometric routing methods from being directly applicable in a network subjected to administrative relationships. The first problem that we identified as the *preference support* one is that of having greedy routing always favouring the shortest possible path (in terms of number of network hops) that might not always be the most preferred one by a network operator (in general a cheaper and reasonably longer path is preferred). The second problem that we qualified as *policy dead-ends* is that of having greedy routing pushing a packet towards a path that it cannot traverse due to the transit rules identified in section 4.2.3. In this chapter we will depict a solution that takes inspiration from the way in which administrative policies are handled in current Internet routing. To do so, let us first study the way administrative policies are handled in the *Border Gateway Protocol* (BGP).

5.1 BGP and administrative policies

In this section we will study the way the current inter-domain routing protocol that is *BGP* deals with administrative relationships between the Internet providers. In this study we will only focus on the *Autonomous Systems* (AS) level of routing and control information and disregard any Internet Protocol (IPv4, IPv6) level details. Hence our study takes place at the level of the AS connectivity graph. More precisely the vertices in the graphs that we treat represent Autonomous

System nodes and not hosts within the AS. Although this approach is a simplification of the administrative routing problem, it allows us for now to focus mainly on the administrative challenge.

In order for an AS network to determine which paths can be taken towards another AS, the AS-level routers in the Internet rely on the BGP protocol that falls into the family of path-vector protocols. According to this protocol, each node participating in the AS-level network announces its presence in the network by sending a *Network Layer Reachability Information* (NLRI) message to all of its neighbours, independently of its administrative relationship with them. Each neighbour AS node receiving such an announcement can then decide to further propagate the advertisement to its own neighbours (except the announcing node) depending on its administrative policies. In the current Internet, the policies upon which this decision is made are mainly the valley-free ones specified in section 4.2.3. Namely, in case the announcer is a customer of the node receiving the announcement, then the receiving node will simply further propagate the presence announcement to all of its neighbours. The motivation is that the announcing node is paying a connectivity service and hence earns the right to be announced to all of the provider's neighbours. In case the announcing node is a provider or a peer, the receiving node will propagate the announcement only to its customer nodes as speculated by the transit rules of section 4.2.3. A node receiving an *NLRI* announcement from a neighbour that is not the originator of the announcement message must also decide whether or not to keep on propagating it. The valley-free policies are also used in this case, however not with regard to the announcement originator (as it might not necessarily be a direct neighbour) but with regard to the relationship with the neighbour from which the announcement was received. As before, in case this neighbour is a customer, the announcement is propagated to all other neighbour nodes. Otherwise, the announcement is further propagated only to the customer nodes. Figure 5.1 shows the propagation of node A's announcement in the network subjected to the depicted administrative relationships. As it can be seen from the figure, node B, when receiving an announcement from its customer node A, further propagates this announcement to all its neighbours. However, when node D receives the announcement from both neighbours B and C decides to propagate it only to its customer F and no to its peer E since both $C \rightarrow D \rightarrow E$ and $B \rightarrow D \rightarrow E$ crossings violate the transit rules defined in section 4.2.3.

As *BGP* is a path-vector protocol, each AS node propagating an

NLRI announcement message includes its identifier in the packet's path trace. These identifiers are in this case the *Autonomous System Numbers* (ASN) that uniquely identify an AS network and are assigned to it by the *Internet Assigned Numbers Authority* (IANA). A node receiving such an announcement learns about a possible path from the announcer to it and hence can obtain a (source-route) path towards the announcer, simply by inverting the sequence of the traversed AS identifiers. Note that such a reverse path from the receiver to the announcer is in all cases a valid path with regard to administrative policies. A simple proof of this fact is that in order for the announcement to reach the current node, it must have traversed a sequence of authorised administrative crossings only. As mentioned in section 4.2.3, such authorized transits are always traversable in both directions.

Following this process a node might learn multiple paths towards a destination. These paths can be learned through different neighbours or also through a single neighbour. Indeed, a neighbour node might in turn have learned multiple paths to a destination and further propagated them to the current node. In the example of figure 5.1, node D learns two different paths towards node A (via B and C) and announces both of them to its customer F. In BGP, whenever a node learns a new path towards a given destination, it must decide whether to update its routing information and to further propagate the new path announcement.

5.1.1 Path selection in BGP

In current implementations, BGP maintains several *Routing Information Bases* (RIB). The main RIB, called the *Local RIB* (Loc-RIB) contains the border router's selected routes towards all possible destinations. All routing decisions in the router are based on the information stored in the Loc-RIB that contains a single entry per-destination. In addition the BGP process stores a separate *Adjacent RIB, Incoming* (Adj-RIB-In) for each one of its neighbours. In this RIB are stored all the NLRIs (i.e. path advertisements) that were received from the neighbour.

In order to add an entry in the Loc-RIB to a given destination, two steps of filtering need to be done. The first filtering step consists in selecting in each Adj-RIB-In, the best entry for the destination. Indeed a neighbour node might advertise several possible paths towards a given destination as the NLRI message might've traversed different network paths to reach it. This selection is in most cases based on the length of

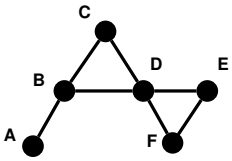


Figure 5.2: Undirected AS level graph corresponding to the relationship graph of figure 5.1

the AS-Path that the NLRI message traversed. Hence the path advertisement with the smallest number of AS hops is usually favoured. All other announcements received from the neighbour and originating from the destination are then deleted from the Adj-RIB-IN base. Given that only one entry per-destination is stored in the Loc-RIB table, the second filtering step consists in selecting the best route to the destination among those offered by several neighbours. In other words, the node needs to select for the given destination, the best route among those stored in several Adj-RIB-In. In current implementations, this selection process is however not based on the length of the route. The strategy chosen instead is that of minimizing the internal cost of packet transits by forwarding them to the AS neighbour offering a cheaper path to the destination. Hence the route selection in this case is subject to local preferences that do not necessarily match with optimality of the route in terms of network distance.

5.1.2 Storage costs in BGP

Given that the Loc-RIB base contains a single entry per destination, the number of entries in a network of N AS nodes can go up to $N-1$ in each table. The number of entries can in some cases be smaller than this amount in case an AS cannot reach some other ASs due to administrative policies. Note that this upper-bound is also valid for the size of the Adj-RIB-In tables as only one entry per destination is stored in this case. Hence a theoretical upper-bound for the amount of routing control data entries stored in a BGP router is $(N-1) * (d+1)$ where d is the number of AS neighbours (or degree) of the AS to which belongs the router. In the current Internet, with more than 39000 ASs participating to the inter-domain network [1], and a maximum connectivity degree going up to 2600 [35], a theoretical upper bound for the number of AS-level BGP RIB entries is above one hundred

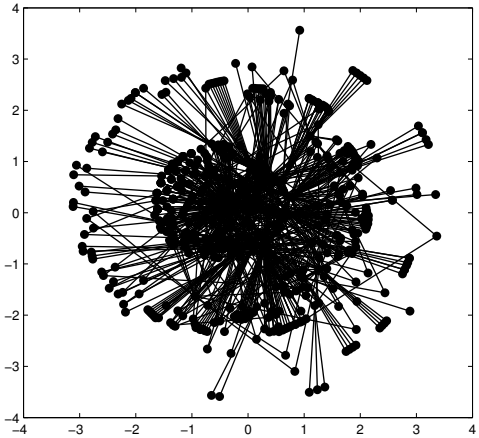


Figure 5.3: 2-D Embedding of an example undirected AS level graph with 500 nodes

million entries (spread over several tables). A major concern in this case is the difficulty of aggregating those entries. Indeed given that the advertised destinations and paths rely on the AS number that is merely a flat identifier, each AS node operating in the Internet must at least store a routing entry for every possible destination. At the IP-level, the situation is even more aggravated by the IP address-space fragmentation. Indeed, some autonomous systems might announce IP address pool composed of several discontinuous prefixes. Hence when operating at the IP level, a single destination AS might be represented by several RIB entries. According to [1], an average BGP RIB base in current BGP routers contains up to 12 Million entries, although only 39000 ASs are participating in the inter-domain network.

5.2 BGP-like approach using Geometric aggregation

In order to prevent greedy geometric routing from forwarding packets towards forbidden paths, we proceed in two steps. In a first step, an embedding of the *undirected AS-level graph* is computed. In this graph,

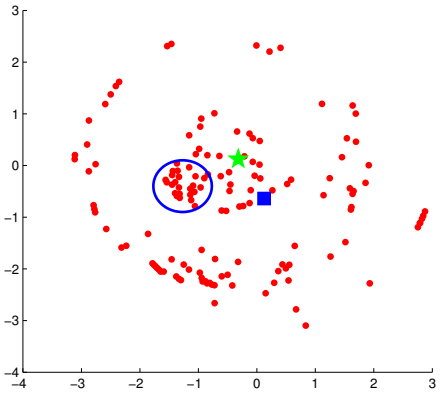


Figure 5.4: Node coordinates accessible by green star node via the blue squared neighbour in the example network of figure 5.3

each vertex represents an Autonomous System and an undirected edge between two vertices represents a communication link between the two ASs, independently of their administrative relationship type. For simplification, in case multiple links between ASs exist, only a single edge is represented in the undirected AS-level graph that we consider. Figure 5.2 shows such an *undirected AS-level*[1] graph for the example network of figure 5.1. We proceed by embedding this undirected graph into a d-dimensional Euclidean space. Note that any of the classical methods for graph embedding could be used at this point. Hence the embedding procedure can be performed in a centralized fashion (by first collecting the AS-level topology), or in a distributed fashion by having a representative within each AS. Note that routing greedily on the coordinates obtained with this embedding will still be subject to the two shortcomings induced by administrative policies.

Therefore, in the second step we propose to make use of the obtained embedding coordinates as aggregatable identifiers rather than for performing greedy routing based on their differences. We rely on a mechanism similar to BGP's path advertisement protocol. In our proposal, each AS node participating to the network announces its presence

[1]Note that for the rest of the document whenever we refer to an *undirected* network, we mean one not subjected to administrative relationships

by sending a message similar to BGP's NLRI to its neighbours. However, instead of indicating its unique AS number in the announcement message, the node indicates its d-dimensional coordinates obtained via the embedding of the undirected AS-level graph. Each node receiving the announcement will then decide whether or not to further propagate it to other neighbours based on its relationship with the announcer. Similar to BGP, in case the announcer is a customer, the announcement is further propagated to all other nodes. In case the announcer is a provider or a peer, the announcement is propagated only to customer neighbours. The same rules apply for a node receiving an announcement from a neighbour that is not the originator of the announcement. However, also similarly to BGP, the announcement is propagated based on the relationship with the relaying neighbour and not the announcer. Hence, if the relaying neighbour is a customer, the announcement is further propagated to all other neighbours. Otherwise, it gets relayed only to customer neighbours.

When relaying an announcement message, each node in our proposal appends its obtained d-dimensional coordinates to the announcement's path vector. A node receiving such an announcement can therefore obtain a path towards the announcer by simply reversing the received path vector.

5.2.1 Control data storage in our approach

Once the exchange of announcement messages stabilized, each node in the network can determine which destinations are available via which neighbour. Note that this is very similar to BGP in which case a node is able to determine, by looking at the Adj-RIB-In base corresponding to a neighbour which AS numbers were advertised by the neighbour. The difference in our case is that the advertised nodes are not flat AS identifiers but rather points in a d-dimensional Euclidean space. Hence, each node in our network can build a *spatial* map of which nodes are accessible via each one of its neighbours.

Figure 5.3 shows a 2-dimensional embedding of an undirected AS-level graph of an example network with five hundred nodes subjected to administrative relationships. The embedding in this figure was obtained by using our implementation of the SMACOF algorithm [12]. Figure 5.4 shows an example result of the NLRI announcements received by the green starred node from its neighbour depicted as a blue square[2],

[2]Note that the two neighbour nodes (green and blue) might appear distant from

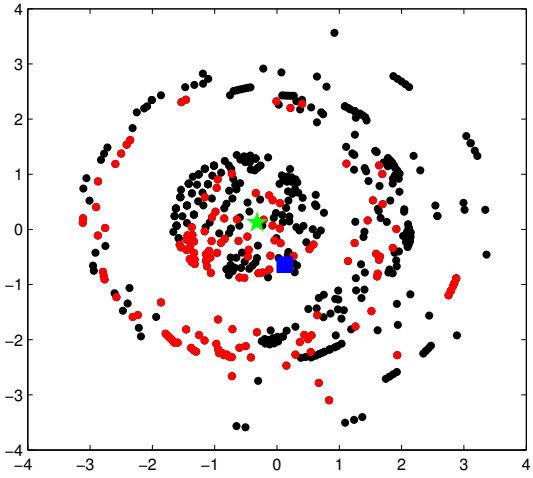


Figure 5.5: Node coordinates (plotted in red) accessible by green starred node via the blue squared neighbour, same as in the example network of figure 5.4. Here, they are contrasted with nodes (plotted in black) accessible via other neighbours

when using the virtual coordinates within the announcements instead of flat identifiers. The advantage of such a representation is that the sub-part of the network that is accessible via a neighbour is expressed in a geometric space (in this case a two-dimensional Euclidean space) as a cloud of points, on which several geometric tools could be applied to compress their representation, as discussed in the following section.

5.2.2 Aggregation of the received announcements

As can be seen from the example in figure 5.4, the set of points accessible via a neighbour is quite spread over the embedding space. However, one can notice the presence of a significant number of subgroups of points that are quite close to each other like the circled set of points in the centre of the figure. Hence a first optimization could be to re-

each other in this figure due to the fact that the embedding is distance-based and the two nodes have a distance of 1 from each other.

duce the amount of control information storage by finding a compact representation for these groups.

For the example circled subset of points in figure 5.4, a straight forward approach would be to compute the convex hull of this cloud and to store in the routing table only the vertices lying on its edges, avoiding by such the storage of the inner-vertices. Later-on, in order for a node to determine whether a destination is accessible via a neighbour or not, it needs simply to compute whether the destination node lies within the polygon (subspace) defined by the convex hull. Note that storing a multidimensional polygon can still turn out to be quite costly as one needs to store the list of multidimensional vertices defining the borders of the polygon. Another way of reducing the accessible node coordinates could then be to use an analytically defined shape to approximate the area's convex hull, such as an ellipsoid. This would reduce the cost of storing the offered network areas (or the cloud of multidimensional points), to the cost of storing a formula. However this method can introduce errors due to wrong approximations of certain polygons. We consider in the following section another form of aggregation of the control information storage through space splitting.

5.2.3 Aggregation through classification

Another approach to the problem of aggregating the RIB information is to consider it as a classification problem. As mentioned previously, once the discovery phase is finished, each network node is able to determine from the advertisements received via each neighbour, a spatial map of the destinations accessible via that neighbour. When combining the list of received announcements from all of its neighbours, a node can get a full map of all the network nodes that it can possibly communicate with. Hence when combining the map of announcements received via a particular neighbour with the full map of all accessible nodes, the node can obtain a spatial map in which the subset of the network coordinates accessible via this neighbour is differentiated from the ones that were announced by other neighbours. Such a map can be seen in figure 5.5. In this case, the red nodes are the ones accessible by the green starred node via the blue squared neighbour, while the black ones are coordinates announced by other neighbours. Determining whether the particular neighbour offers an access to a destination can then be translated into determining to which of the two categories do the destination coordinates belong. Hence routing when using this spatial map

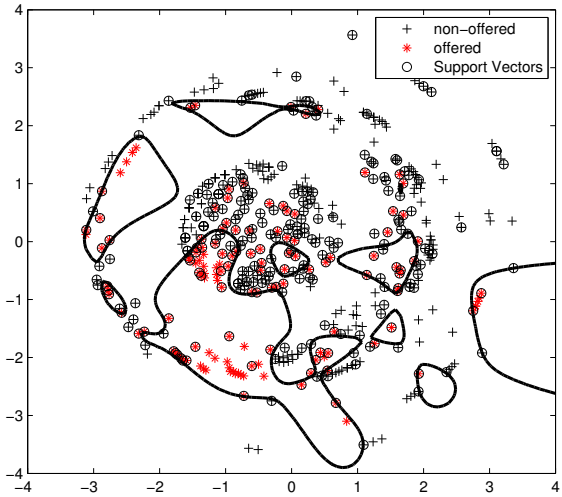


Figure 5.6: Decision surface computed by SVM for the example of figure 5.5 using 277 support vectors

can then be thought of as a classification problem : i.e. given a destination's coordinates, determine to which of the two classes *accessible* or *non-accessible* it belongs. The advantage of seeing the problem under this angle is that it allows us to compress the routing control data required to the mere storage of the classifier.

For the rest of this chapter we will be investigating the potential use of classifiers as a mean to compress the network coordinates received

In the next section, we will give a brief overview of classification methods and the traditional goals and challenges in this field. We encourage the reader familiar with classification literature to proceed to section 5.2.5, where we differentiate our usage of classifiers from the classical one presented below.

5.2.4 Quick overview of classifiers

In the classification problem, one is provided with a data set containing a sample of multidimensional points along with a discrete label data indicating to which class each data point belongs. In the binary classi-

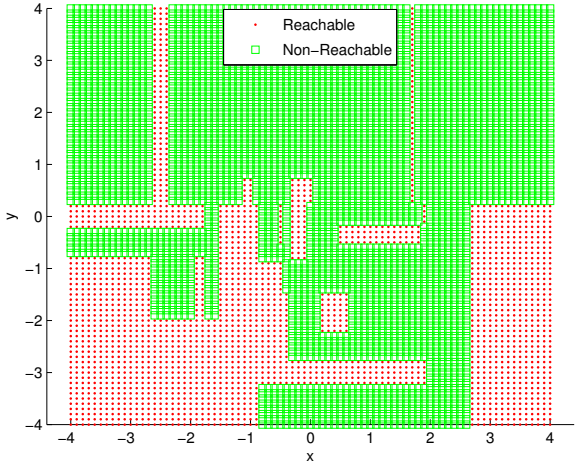


Figure 5.7: Class separation computed by the decision tree for the case of Figure 5.5

fication problem, the number of classes in the data set is two. The role of a classifier then is to find a separation methods between the multidimensional data points belonging to different classes. The goal is to be able, when encountering a new non-labelled data point, to attribute it to one of the two (or more) classes. The provided data set on which the classifier is initially computed is referred to as the *training data* and is assumed to be a sample from the possible existing data points. Since the classical use of classifiers is to perform a prediction on the class of a given data point, great importance is given to the generalization power of the classifier. A good classifier must not necessarily fit exactly to the training data set as this might contain incoherences, errors, or outliers (extremely rare cases). In that case the classifier would fit exactly to a skewed view of reality and therefore perform poorly with real-world data, although it had perfect predictions with the training data set. This problem is commonly known as *over-fitting*.

Classifiers can be differentiated based on whether they rely on the probability distributions of the data samples or not. In the first category, for each class, a probability density function is assumed to indicate for every possible data value the probability that the data point belongs

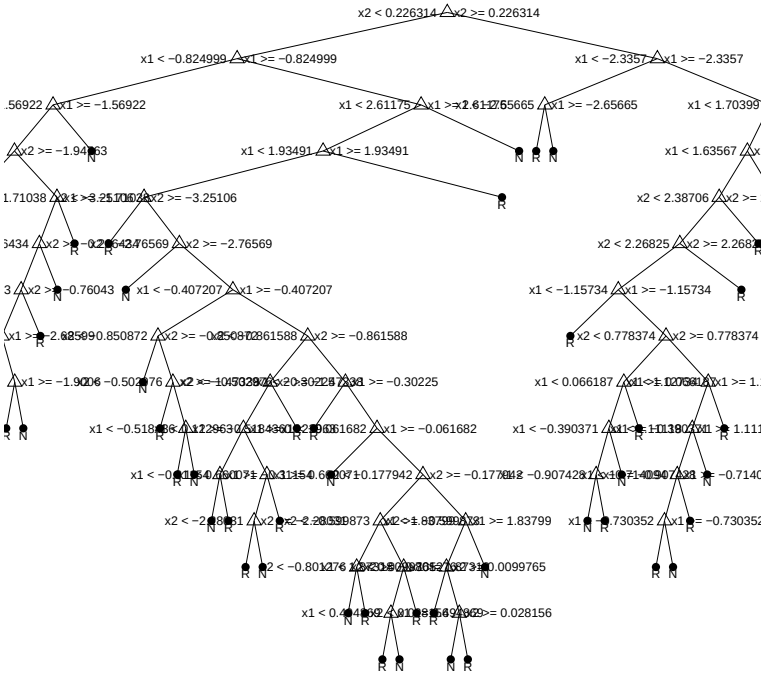


Figure 5.8: Partial view of the decision tree generating the class separation in figure 5.7, exhibiting a maximum depth equal to 14. R value at leaf means the coordinates is reachable and N non-reachable

to the class. Hence when provided with an unknown data point, this family of classifiers can provide levels of certainty about the association of the point with a given class.

A more categoric family of classifiers simply tries to find a separa-

tion method between the multidimensional points. Among these, the *Perceptron*[93] is probably the most simple approach. The goal of the perceptron method is to draw a separation line (or hyper-plane in the case of multidimensional data) that splits the data points belonging to the two classes. Ideally, the goal is to have all data points belonging to the first class lying on one side of the hyperplane and all data points of the second class on the other side. In the classification terminology this hyperplane is referred to as the *decision surface* as it is the only element needed to perform the classification decision of a newly encountered point. One has simply to verify on which side of the hyperplane the new point resides in order to determine the class to which it belongs. One major drawback of the perceptron, is its poor generalization capability, as multiple separation hyperplanes might be possible (in case the classes are linearly separable) and hence any of these lines might be selected regardless of its generalization properties. *Linear Support Vector Machine (SVM)* [24] classifiers solve this problem by computing the separation hyperplane that maximizes the margin between the training data points belonging to the two classes. The intuition being to leave the most possible space between the decision surface and the training data in order to account for real data points that fall in this region and hence to obtain better prediction results.

However not all training data is linearly separable as can be seen in the example of figure 5.5. For such cases, both the perceptron and the SVM approaches proceed by first transforming the given data points into a higher dimensional space in which they would be linearly separable. A linear decision surface is then computed in the high-dimensional space. When translated back into the initial training data space, this decision surface will describe a non-linear trajectory. In the SVM case, this transform is done via the *kernel method*. In the Perceptron, the transformation into the higher dimensional space is achieved by inserting additional layers between the input and the output layers. The possible curvatures taken by the decision surface depend on the number of additional layers. With one additional layer, problems with convex decision surfaces can be solved, while with two layers more complex (possibly any) decision surfaces can be learned. Also in the non-linear case, the main difference between the perceptron and the SVM approaches remain in the better generalization capability of the SVM due to its margin maximization constraint. Figure 5.6, shows the non-linear separation line obtained by using the Support Vector Machine method.

Another family of classifiers performs the class separation task by

recursively splitting the data samples according to a value of one of the input attributes. Typically the attribute and the value are chosen so as to discriminate at best between the two classes.

In the case where the input values are d-dimensional (Euclidean) variables, the decision surface encoded by the data split can be represented by a $d-1$-dimensional hyperplane that is orthogonal to the dimension of the split and intersecting it on the point corresponding to the separation value. Ideally, all the data points lying on one side of the separation hyperplane should belong to the same class. If this is not the case, then the space splitting procedure can be recursively repeated on each of the two subspaces delimited by the separation hyperplane(s). In fact, the space splitting procedure can be recursively repeated until all data points are correctly classified. This classification method is known as the *Decision Tree* method, as the different separation steps can be represented as a tree as shown in figure 5.8. This figure shows the decision tree obtained when applying the recursive space splitting method to the two-classes data set of Figure 5.5.

The inner-nodes of this tree (also denoted as branches), indicate both the parameter (or dimension in case of d-dimensional data) and the value at which the split is to occur. The leaf nodes indicate the classification decision that is computed by the tree. When classifying an unknown data point, according to the decision tree, one must simply traverse the tree following the branches that correspond to the data values. The class attributed to the data point is then the value of the leaf node that is reached. Figure 5.7 shows the class attributions (or in other words the region splits) to the two-dimensional points when applying the decision tree of figure 5.8.

5.2.5 Using classifiers to store routing data

Given a dataset similar to the one depicted in figure 5.5, we can train a classifier that allows to efficiently discriminate between the offered and non-offered destination coordinates. In case all data classification is correctly done we could then store only the necessary data to perform the classification operation and hence dispose of the rest of the data. To give an example, let us assume that a node A received from a neighbour node B a list of reachable destination coordinates and that when combining this list with those received from other neighbours than B, the node can construct a dataset similar to that of figure 5.5. Let us also assume that in the case of this dataset, the reachable destination

coordinates can be linearly separated from the non-reachable ones (i.e. we can draw a hyperplane on one side of which the reachable destination coordinates lie and the non-reachable on the other). In such a case, node A can reduce the routing information received from its neighbour B, to the mere information required to perform the classification task. In this case, all we need to store is a d-dimensional vector $\boldsymbol{\omega}$ (where d is the dimensionality of the destination coordinates) and a scalar value ω_0 that together define the separation hyperplane's equation $\boldsymbol{\omega^T} x + \omega_0 = 0$. Hence all the destination coordinates received via neighbour B can then be deleted since in order to determine if a destination address belongs to the reachable group via B or not, node A can simply check on which side of the hyperplane the destination point lies.

We would like to highlight at this point that our intention of using classifiers strongly differs from the classical use of these methods. Indeed, in the regular usage, and as mentioned in the previous section, one is mainly interested in the generalization capabilities of classifiers in order to perform inference tasks on new data points. In our case however, our main purpose is to use these methods as a compression tool for our received node coordinates. Our belief is that, by extracting the features of the network at hand, such as network connectivity, and representing it in a feature space, the decision surface computed by the classifier should be *minimal*, as the classifier would *learn* the hidden rules that guided the service differentiation in the network and thus the routing decision process. Also, in addition to the compression of the neighbour's routing table, classification methods also present the advantage of speeding up the forwarding procedure. Indeed, in the previous example case of nodes A and B, node A would have to perform a simple linear computation (to determine the sign of $\boldsymbol{\omega^T x_d} + \omega_0$ where $\boldsymbol{x_d}$ is the destination's coordinates vector) in order to determine if the destination is reachable via node B or not. Hence the forwarding operation's (decision computation) cost is the same minimal cost for all the destination coordinates.

When selecting the appropriate classification method for our destination reachability data, two properties are to be optimized. These are the size of the data required to perform the classification operation as well as the speed of the classification in order not to delay (and hopefully speed-up) the forwarding procedure. Note that in our particular usage of classifiers we do not allow for classification errors since they would induce false routing information. Hence when evaluating the amount of data required for performing the classification (the size of the classi-

fier), we also take into account the amount of misclassified destination coordinates. The goal being to store those coordinates in addition to the classifier data-structure in order to ensure that the obtained classification decision is correct.

In the classification methods listed in the previous section, the ones relying on assumptions on the probability distributions of the data are in fact not well suited for our purposes since they are mainly tailored to perform inference on new data. Indeed, in those methods, the provided dataset is assumed to be only a subset of the real-world data which is not the case for our coordinates reachability dataset. Also, in our case the membership of destination coordinates to the reachable (or non-reachable) class is dependant on the cascade of administrative relationships in the network between the node and the destination and hence it might be difficult to estimate a probability distribution function. Note that one could consider applying unsupervised learning techniques such as *Gaussian Mixture Models* to approximate the probability distribution of the reachable and non-reachable classes however this bares a big similarity with the reachable area's shape approximation approach via more categoric methods such as decision trees and SVM, while requiring an additional decision step based on the probability estimations.

Linear classification methods are also not well suited for our purposes as they induce large data storage. Indeed, as can be seen on figure 5.5, our coordinates reachability dataset is hardly linearly separable. Hence relying on a linear classification method would induce a large amount of classification errors, and since we do not allow for such errors (by storing misclassified points), this would inflate the size of data required to perform the classification operation correctly.

By design, the Support Vector Machine classification method was tailored to enhance inference (prediction) decisions by maximizing the margin between the two different classes (in both linear and non-linear cases). Such a behaviour has the disadvantage of increasing classification errors by having some points lying on the wrong side of the decision surface and within the margin, thus inflating the amount of data required to perform reachability classification correctly. As we are not concerned by prediction capabilities, we can tune the SVM parameters in order to fit as close as possible to the dataset as demonstrated by the decision surface in figure 5.6 hence reducing the number on misclassified points. However, in order to operate, the SVM classification method requires storing a subset of data points that are close to the decision surface that are known as the *support vectors*. As can be seen

from the example in figure 5.6, the number of such data points can be rather large (277 support vectors in the case of 500 nodes). Not only would such a large number increase the size of the classifier data structure, but in the case of SVM, it would also slow down the classification procedure. Indeed in order to determine to which class a data point belongs, the SVM method must check the sign of a value, the computation of which involves all the support vectors data points. Hence the bigger the set of support vectors, the longer the computation. For these two reasons, we did not consider using the SVM method for our destination classification task.

In multi-layer perceptron classification method, we are only required to store the parameters of each of the perceptron nodes (neurons). These parameters consist of a multidimensional vector representing the gradient of the hyperplane defined by the perceptron node as well as its bias value. Hence in case a relatively small multi-layer perceptron network is used, the storage required for the classification method as well as the processing time would be minimal. However, due to their design, multilayer perceptron networks are hard to tune towards a desired result minimizing the amount of misclassified points. Therefore, without eliminating the possible use of the multi-layer perceptron (and other generic forms of neural networks), we would rather for the time being focus on more accessible types of classifiers.

In the case of decision trees, the storage cost of the classification method depends mainly on the number of *branches* (inner nodes) of the tree since the leaf nodes simply encode a binary value. The branches instead, contain a floating point value indicating at which point the splitting hyperplane should intersect the dimension of the split. Hence each branch node is in fact encoding one single component of the multidimensional destination coordinates, along with the dimension index at which the split is to occur. Note that in the case of the classification tree method, a perfect fit to the data is always possible. Indeed, one can simply recursively repeat the space-splitting method until each data point is correctly classified without any concern about over-fitting the data (since we do not need to perform any predictions, over-fitting is not a concern). However, some bad positioned data points, such as a reachable coordinate completely surrounded by non reachable ones, might require multiple additional space splits in order to be correctly classified, hence inflating the size of the tree. Therefore, partial tree pruning might reveal beneficial, as storing a few additional error coordinates could induce less storage than a classification tree perfectly

fitting the dataset.

The decision tree method also offers the advantage of *indexing* the destination coordinates dataset and hence speeding-up the forwarding process. In the example decision tree of figure 5.8, the *depth* of the tree, i.e. the number of inner-nodes on the longest path from the root to a leaf is equal to 13, meaning that in the worst case, it would take up to 13 comparison instructions in order for the classification tree to reach a decision on whether some destination coordinates are reachable via a neighbour or not. This performs much better than iterative comparisons on the list of received coordinates that could take up to N-1 comparisons, and still comparable with the cost of a binary search of $O(\log N)$.

5.2.6 How to compute decision trees

In order to determine on which attribute and value to perform a split, the recursive space splitting approach can rely on multiple measures, the most common of which is the *Information Gain*. This measure relies itself on the notion of *entropy* defined by Shannon that measures the level of purity (uncertainty) associated with a set. The entropy of a set S_v is defined as follows :

$$H\left(S_v\right) = -\sum_{i}^{M} P\left(c_i|S_v\right) \log_2\left(P\left(c_i|S_v\right)\right)$$

where M is the number of classes in the dataset and $P\left(c_i|S_v\right)$ is the probability of class c_i given the set S_v. Given a sample dataset S_v, the probability of a class c_i given that set (i.e. $P\left(c_i|S_v\right)$) is simply equal to the number of element within S_v belonging to class c_i, divided by the total number of elements of S_v. When separating the elements of a set S according to their value with regard to an attribute a, the information gain over this attribute can then be measured as follows :

$$IG\left(a\right) = H\left(S\right) - \sum_{v} \frac{S_v}{S} H\left(S_v\right)$$

where S is the set of points before the split, and S_v are the subsets resulting from the partition. The difference between the entropy of the set before split and the normalized sum of entropies of the resulting subsets allows to measure the amount of information gained by the split, or in other words, the level of set purity that was gained. Hence in order to determine on which attribute to perform a split, the recursive space

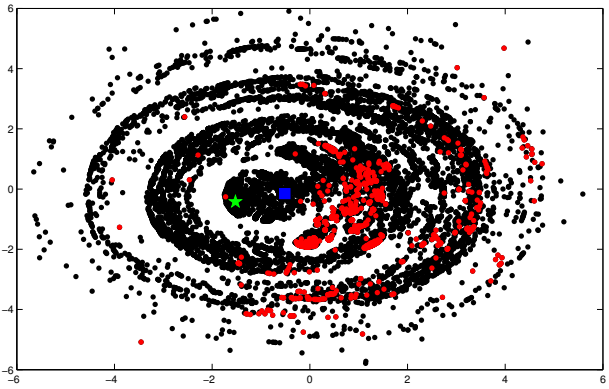


Figure 5.9: 2-dimensional embedding of CAIDA 2004 graph (embedding stress = 0.33) and an example of destination coordinates offered by blue squared node to neighbour green starred node. The offered destinations are plotted in red

partitioning algorithm must evaluate for each of the possible attributes (and for each of the possible values of every attribute), the amount of information gained by performing a partition at that attribute and value and select the combination with the maximum gain.

In our use of the recursive partitioning algorithm, we stop the development of the decision tree (i.e. we stop the splitting recursion), when the number of data points associated with a tree branch node is smaller than a constant threshold (in our case 5 data points). This will avoid defining multiple splits to correctly identify some outlier points, thus resulting in more storage than is necessary to store the point.

5.2.7 Representing routing information using decision trees

For the reasons depicted above we decided to evaluate the performances of our routing information representation using decision trees classification. We applied the recursive space splitting algorithm on the embedding coordinates of the Autonomous Systems connectivity graph as it was collected by CAIDA [2] in *2004*.

In the CAIDA dataset are listed 16300 autonomous system nodes

along with their connection information labelled with the administrative relationship type : Peer-Peer, Customer-Provider, Sibling-Sibling. We simplify the dataset by pre-processing it in order to get rid of the sibling relationship by collapsing all sibling components into a single vertex summarizing all the relationships to other network nodes. This reduces the number of vertices in the graph to 16156. We then perform a centralized multidimensional scaling embedding using the SMACOF [12] iterative majorization technique in order to embed the AS graph into a two-dimensional Euclidean space, based on the shortest-path distance matrix. Note that the shortest path is computed independently of the link's administrative types and hence these distances do not necessarily correspond to the distances subjected to administrative routing. The Embedding coordinates of the AS graph can be seen on figure 5.9. In order to determine for each node which destination coordinates are offered by neighbour nodes, we simulate the path vector announcement protocol using the embedding coordinates in the announcement instead of regular node identifiers. Hence, similar to the example of figure 5.5, a spatial map in which the set of reachable destinations can be differentiated from the non reachable ones can be inferred as in figure 5.9.

Each node in our simulation then builds a decision tree to represent the destination coordinates received from its neighbour. Hence there are 2 decision trees built per network link.

The recursive space-splitting approach proceeds by maximizing the information gain value for every computed split and stopping the growth of the decision tree when the size of a non-homogeneous data point subset is lower than 5 points. We then evaluate the storage required for each computed decision tree by summing the number of its branch nodes along with the classification errors. In order to measure the classification error, we simply resubmit the network coordinates data labelled with the class membership (offered/non-offered) of each data point (in classical classification terminology, this is equivalent to resubmitting the training data to evaluate the training fitness).

Figure 5.10 shows the storage amount (in number of entries) required for every network link in both directions, given that the announced offered destinations from a node to its neighbour is different from the set of points announced by the neighbour (hence on the X axis of figure 5.10 there are double as many values as there are network links). A striking observation in this figure is the unfairness of required storage between different links in the network, namely that there is a minority of network links (and hence nodes) having to bear with very large amounts

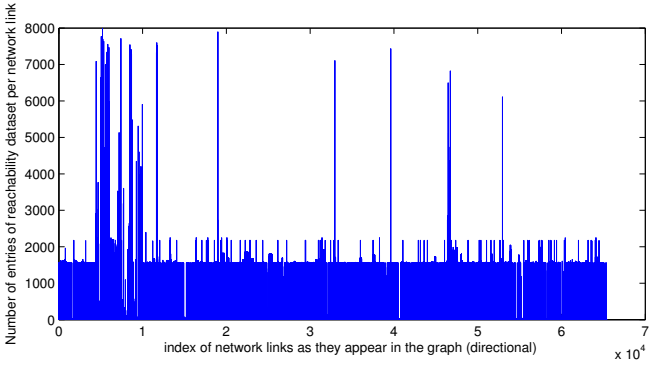


Figure 5.10: Number of table entries according to the reachability dataset of CAIDA 2004 AS graph vs the network link index as it appears in the graph file. Note that the maximum number of entries never exceeds the half of the number of AS nodes (16156), since in that case one can simply store the complement subset (i.e. the non-offered destination coordinates instead of the offered ones)

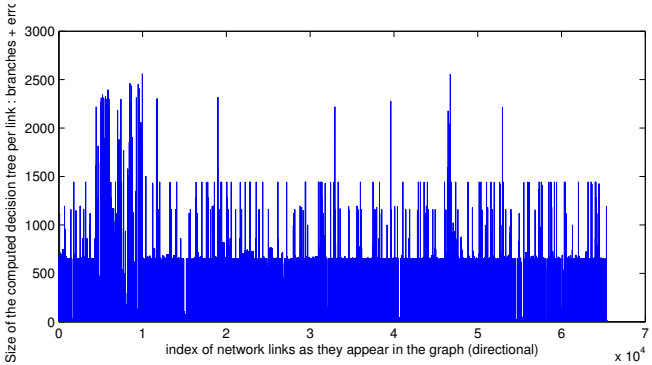


Figure 5.11: Size of the computed decision tree per link in the CAIDA 2004 dataset

of data compared to the rest of the links.

Figure 5.11 shows the size of the computed decision tree per network link for the same network graph. Notice that the unfairness between the minority of central links and the rest is still visible, however the discrepancy is greatly reduced.

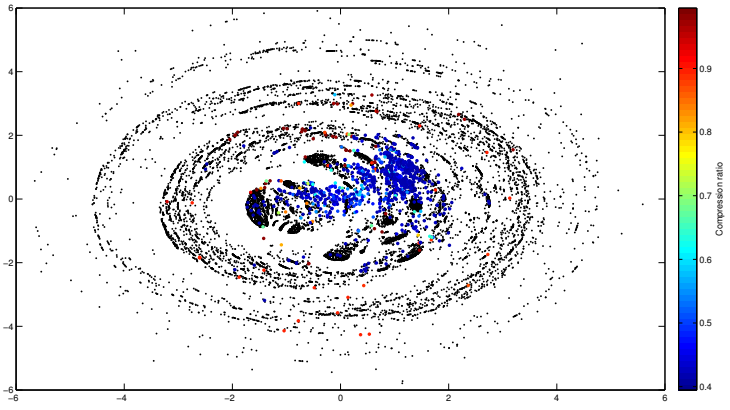


Figure 5.12: Compression ratio achieved when using the decision tree method with regard to the embedding position. One can see that the central (core) nodes are the ones profiting at most from the usage of the classifier.

In order to better visualize the results of figures 5.10 and 5.11, figure 5.12 displays the compression ratio achieved when using the decision tree classification method as a function of the node coordinates. In order to obtain the compression ratio value for a given network node, we simply divide the sum of all the storage size of the trees computed for the node's neighbours by the sum of the storage required without compression (i.e. the values in figure 5.10) as follows:

$$Ratio\,(n) = \frac{\sum_i \text{sizeDecisionTree}\,(n,i)}{\sum_i \text{sizeOfferedArea}\,(n,i)}$$

where i iterates on the neighbours of node n and sizeDecisionTree(n,i) returns the size of the computed decision tree for the reachability dataset announced by i to n (consisting of the number of branch nodes along with the amount of errors) and sizeOfferedArea(n,i) returns the number of entries required to store the destination coordinates announced by neighbour i to node n.

Note that not all the nodes do necessarily benefit from the decision tree usage. In fact, the nodes having to store a very small amount of offered destination coordinates might end up storing more data when

using the decision trees as many space splits might be required to isolate the few spread individual offered coordinates. Given the small size of the received destination coordinates, the concerned nodes can simply rely on the storage of the received coordinates list as is as it does not incur a heavy storage load.

However, one can notice in figure 5.12 that the nodes profiting at most from the usage of the decision tree method to represent the reachability data are in fact those at the centre of the network, namely the main tier nodes in the AS-graph. These are the nodes having to store the peak values of figures 5.10 and 5.11. As can be seen from the figure, the compression ratio for these nodes can go down to 0.4 meaning that when using the decision tree classifier a node can get rid of 60% of the reachability data without introducing any routing errors.

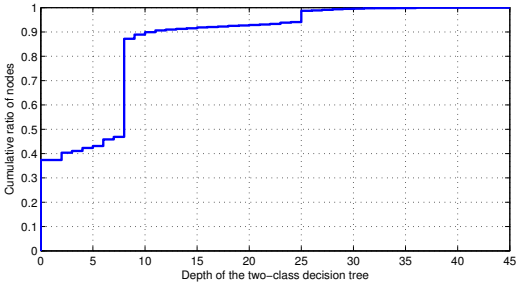


Figure 5.13: Cumulative distribution of the depth of the computed decision tree per network link

Figure 5.13 shows the cumulative distribution of the depth of the decision trees computed for every network link. By *depth* of the tree, we mean the longest path from the root to a leaf node. This value is important since it represents the worst case scenario for the classification, i.e. the longest series of comparison instructions before the classifier can reach a decision on whether the input destination coordinates are offered by a neighbour node or not. As can be seen from the figure, the largest depth value of the computed decision trees is 45, meaning that in the worst case, it would take a node at maximum 45 comparison instructions in order to determine if a destination coordinate is accessible via a neighbour (instead of parsing a list of 16155 entries) hence guaranteeing a fast decision. A more important fact, visible in the depth distribution in figure 5.13 is that for almost 90% of the network links the

Figure 5.14: Example unidimensional map (bar-code) of the destination nodes offered by a neighbour in the CAIDA 2004 network : node ids at which a bar is displayed are those announced by the neighbour

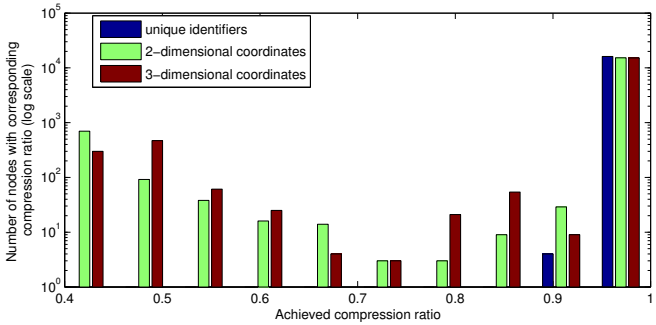


Figure 5.15: Compression Ratio distribution: comparison between identifiers and 2 and 3-dimensional network coordinates

computed decision tree has a depth inferior or equal to 10 meaning that a striking majority of the inter-domain routing decisions in the CAIDA 2004 graph can be performed in less than 10 comparison instructions.

Note that the achieved decision speed is mainly due to the *information gain* optimization of the split. Indeed, when constructing the decision tree, the recursive splitting algorithm optimizes the purity of the resulting subsets that also corresponds to the amount of information gained by performing the split. This optimization is naturally also valid for the operation of the decision tree and guarantees an optimal series of decisions (comparisons) in order to classify the input data in the fastest possible way.

Also, we would like to point out that the achieved compression ratio and the minimal tree depth are mainly due to the use of the embedded network coordinates. Indeed, although these coordinates cannot be used for a successful greedy routing, they are still encoding topological features of the network graph that are then exploited by our classification process as demonstrated above. In order to empirically prove this

we will in the following perform the same classification procedure when using flat node identifiers instead of the two-dimensional coordinates. The identifiers attributed to the network nodes in this case are flat succinct identifiers that are randomly attributed to nodes without carrying any proximity information. In this case, we simply perform the classical inter-domain path advertisement approach using node identifiers in the announcements.

Note that this procedure also allows us to deduce a unidimensional "spatial" map of the nodes offered access to by a neighbour as in figure 5.14. In this figure, the horizontal axis represents the node identifiers and a vertical black bar is drawn on the index of the nodes that have been announced by the neighbour. Hence, the same classification method used above could be applied to the example unidimensional case of figure 5.14 in order to find a separation "surface" that allows us to represent the unidimensional data in a compacter way.

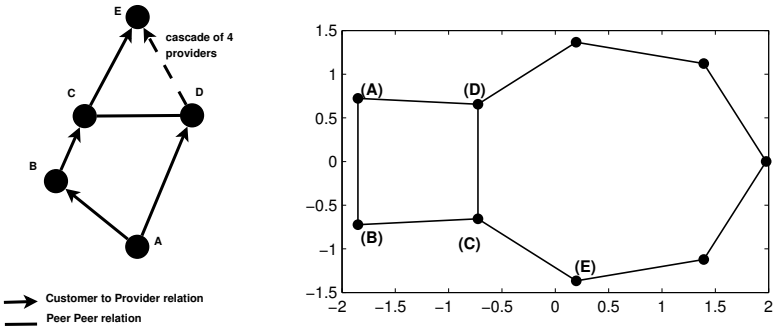
Table 5.1: Compression Ratio Distribution when using classifier on identifier data

Compression Ratio	Number of nodes
0.89	2
0.90 to 0.98	6
0.99	16148

Table 5.1 shows the compression ratio distribution obtained when using the decision tree classifier on the one-dimensional reachability data based on unique identifiers attributed to nodes. As can be seen from the table, the compression ratios are far from the performances achieved when applying the decision tree classification on the reachability dataset using spatial coordinates. This is due to the fact that the embedded coordinates allow for a better separation between the offered and the non-offered destination coordinates since it is very likely that a subset of close-by nodes belong to the same class.

However, the distribution of offered destination flat identifiers is "spatially" highly fragmented as can be seen in figure 5.14, thus requiring as many spatial splits as there are data points.

Effect of dimensionality when using decision-tree classifier In figure 5.15 we give in a logarithmic scale, a histogram that compares the distributions of compression levels achieved for numbers of nodes,



(a) administrative relationships

(b) Metric embedding based on the non-administrative distances

Figure 5.16: The non-administrative distances can in some cases strongly under-estimate the real distance of the administratively authorized path

when using unstructured flat identifiers, 2-dimensional embedded coordinates and 3-dimensional embedded coordinates. This histogram reveals 2 important things. On one hand the superiority of employing virtual geometric coordinates against an unstructured identifier space that is discussed in the previous section. On the other hand the minimal improvement of increasing the dimensionality of the embedding (Kruskal's stress 1 error parameter was 0.25 for the 3D case versus 0.33 for the 2D case). This suggests that a 2D embedding is sufficient to obtain a good discrimination between offered and non-offered destination coordinates, without the need for higher-dimensional node coordinates that could increase the decision tree storage costs.

5.3 Multiple Classes and Distance Function Regression

When using the decision tree classification method in the previous section, an important element of the reachability data was not taken into consideration. This element consists in the network distance value towards reachable destination coordinates.

Indeed, the previously used reachability data was binary, simply indicating whether a destination's coordinates is offered or not. Hence in

a situation where a node determined that a destination is accessible via two or more of its neighbours, the node is unable to determine which of the two is closer to the destination. One could imagine that the Euclidean distance between the neighbour and the destination could give a good estimate of the network distance. However, given that the embedding coordinates that we use are obtained when considering the shortest distance matrix between network nodes **without considering administrative relationships**, the obtained embedding distances might be a poor estimation.

To give an example, consider the case of Figure 5.16 where both providers B and D offer access to destination E to node A. When considering the non-administrative network distance towards destination E, both B and D have the same distance (equal to 2 through C) that would also be reflected in their attributed coordinates distance to node E's coordinates as shown in figure 5.16b. In such a situation, when relying simply on the binary reachability data and the Euclidean distance between the embedding coordinates, node A might be tempted to forward packets destined to E to neighbour D thinking that it is at a similar distance to destination as neighbour B (D might even appear closer to E due to embedding distortions), while in reality the authorized path is much longer as can be seen on figure 5.16 (note that the path $D \rightarrow C \rightarrow E$ violates the valley-free constraints).

Hence, in order to allow node A to better decide on the next hop to the destination, the administrative distance, retrievable from the number of hops in the received announcement, need also to be reflected in the reachability data. This can be easily achieved by extending the previous classification solution to perform a multi-class decision instead of a binary one. Instead of considering a model in which destination coordinates can belong to only one of two classes (reachable or non-reachable), we consider a model where it can be part of one of many classes corresponding to the shortest distances along administratively allowed paths to a destination's coordinates. In this model, there is a finite number of classes equal to $d+1$ where d is the network diameter when considering administratively allowed paths. The additional class (+1) is the one attributed to non-offered destination coordinates. Figure 5.17 shows the same reachability data of figure 5.5 with each data point labelled with a class corresponding its administrative distance to the destination or with the non-reachable class label in case the neighbour node did not announce its coordinates. As can be seen from the figure, the number of obtained classes is fairly limited as the most administratively distant

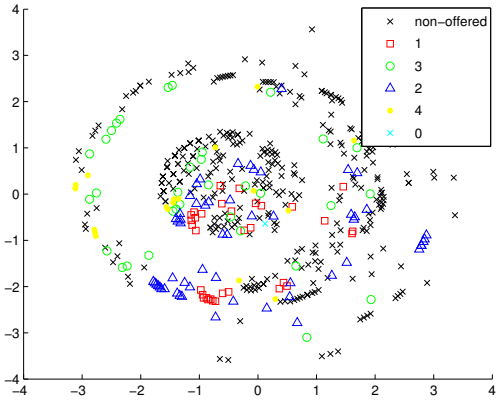


Figure 5.17: Destination coordinates labelled with a class corresponding to their distances to neighbour, or to the non-reachable class in case there is no administratively authorized path towards them

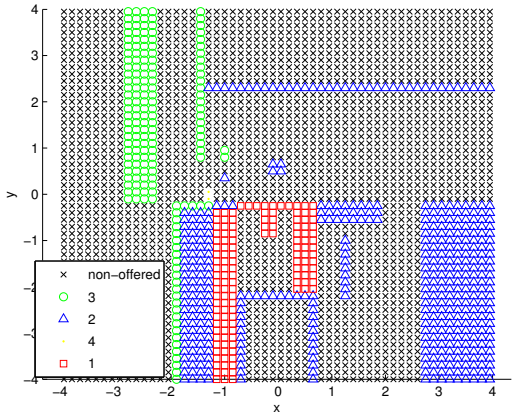


Figure 5.18: Output of multi-class decision tree on the dataset of figure 5.17

nodes from the neighbour are at a distance 4. The same classification method can then be applied on this dataset trying to separate at best the offered nodes with the same distance from offered destinations with

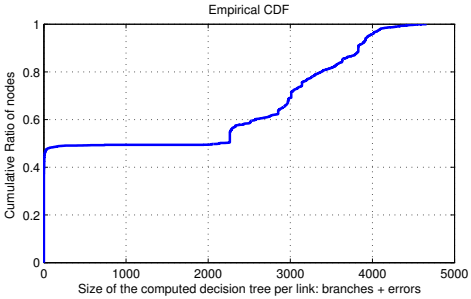


Figure 5.19: Cumulative Distribution of the size of computed multi-class decision trees

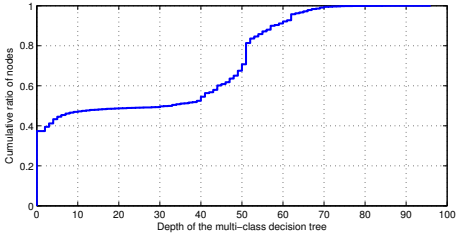


Figure 5.20: Cumulative Distribution of the depth of the computed multi-class decision trees

a different distance and from the non-offered destination coordinates. Figure 5.18 shows the result of such a classification on the dataset of figure 5.5, when using the decision tree method. The same information gain approach is also used in this case to construct the classification tree.

We applied this classification approach to the CAIDA 2004 autonomous systems graph. Figure 5.19 shows the obtained cumulative distribution of tree sizes results, where the computed size accounts for the number of branch nodes plus the classification errors. One can notice that the maximum tree size almost doubled when compared to the binary classes case. However and as shown by the cumulative distribution plot, a large majority of the links result in decision trees with a size smaller than 4000 meaning that the cost of the decision tree is lower than that of storing a distance vector of 4000 elements. Considering that

the computed multi-class decision trees (with the errors compensation) allow us to correctly compute the distances to all the network nodes, this means that we are able to restore more than 16000 distance values by using only 4000 entries and hence we have a compression ratio similar to that of the binary classes case.

Figure 5.20 shows the obtained depth of the multi-class decision trees. In this figure one can see that the worse case tree depths also double compared to the previous binary case. However as shown by the cumulative distribution curve, almost 80% of the links incur a worst classification depth of a little more than 50 instructions which is satisfactory considering that the plotted values are only the extreme cases.

Note that the multiple classification when relying on a decision tree can result in classification errors either indicating that a non-reachable node is reachable (or vice-versa), or indicating that a reachable node is at a wrong distance from the neighbour. Therefore, in case we target a lossless compression of the routing data, a compensation for the errors would be necessary by additionally storing them. This error storage was taken into account for the storages costs reported in figure 5.19 as it was the case for the binary dataset.

5.3.1 Relation to distance metric learning

In the multiple classification method used above, each node was provided with a dataset consisting of d-dimensional coordinates and their corresponding distance values. By using a classification tool, each node then found a method for *approximating* the distance value when given a destination's coordinates as input.

Figure 5.21 shows a plot of the received reachability dataset. In this figure, the blue points are data points transmitted by the neighbour to the node, consisting of a 3-dimensional point where the first two dimensions are the network coordinates and the third dimension is the communicated distance value. In this figure, all non-reachable destination coordinates have been attributed a high constant value for the distance (bigger than the network diameter and equal to 10 in this case), in order to reflect the impossibility of reaching these destinations. The surface, on which all the data points lie, is constructed by performing an interpolation between the data points thus allowing to better visualize the distance data variations. Figure 5.21 is in fact a visual explanation for the failure of greedy routing based on the Euclidean distance in a network subject to administrative relationships.

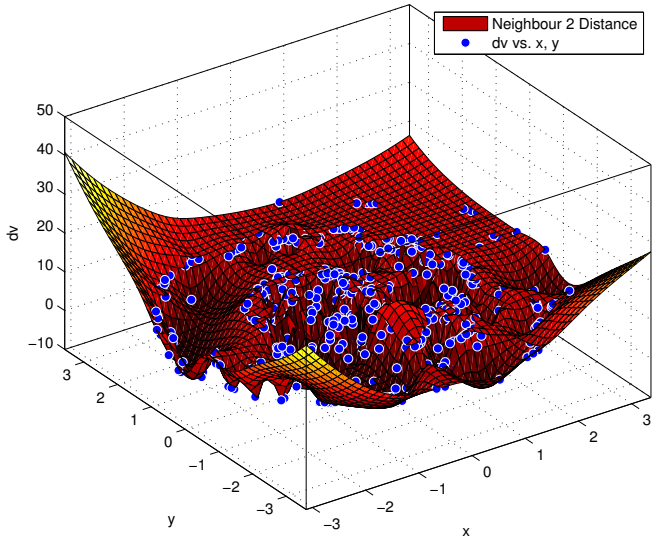


Figure 5.21: Interpolated "distance function" based on the reachability dataset of figure 5.17

Indeed, the interpolated surface strongly differs from the output of the Euclidean distance function represented in Figure 5.22, that would describe a *cone* centred at the neighbour's coordinates and in which the distances increase linearly the further we get from the neighbour node. Instead, the transmitted dataset's surface is very irregular describing sudden peaks in the distance value at the non-reachable point coordinates. Therefore another way of approaching the problem of greedy routing under administrative relationships is to consider it as a distance function regression, or *distance metric learning* [118, 119] one, in which a node after having received the reachability dataset from its neighbour, must infer a distance function that fits the received dataset better than the implicit Euclidean one. Note that our multiple classes decision tree can be compared to a regression tree fitting the transmitted distance vector data. However approaching the regression problem under the multiple classification angle has the advantage of reducing the size of the generated decision tree, as the number of obtained classes is minimal

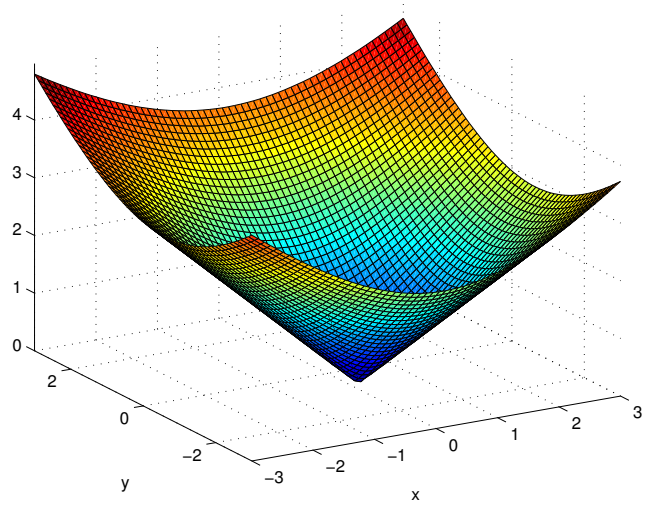


Figure 5.22: Euclidean distance function from the neighbour node in figure 5.5 (blue node)

thus incurring a minimal value storage cost at the tree leaves. In the regression tree case however, the generated values at the leaf nodes would be very diverse thus reducing the compression benefit of our method.

5.4 Comparison with conventional routing and forwarding table structuring

In the following, we position the above presented work with regard to the common techniques of structuring and management of routing and more precisely forwarding tables in the current Internet.

One of the most wide spread data-structures in today's router devices is the *trie* with its many variants, binary, multi-bit, *Patricia*, *Lulea* etc. [113]. Trie data structures are highly efficient when performing longest prefix matching tasks. The goal of this task is to find the best matching routing entry in a context where a hierarchical addressing scheme, similar to the Internet IP addressing, is assumed and routing

information entries are aggregated according to this scheme. Note that the necessity of performing longest prefix matching in this context comes from the fact that exceptions to some larger range prefix rules may be needed as for example having an entry "$1.* \rightarrow interface\ A$" and an additional forwarding rule "$1.2.3.* \rightarrow interface\ B$" (where the asterisk $*$ stands for any values). Therefore, when handling a packet label, one must verify that the rule with the longest prefix is matched as a label such as 1.2.3.4 would be matching both prefixes. In its most simple form that is the binary trie, the search proceeds by examining bit by bit the label to be matched. Starting from the root of the trie, the search simply moves down the trie structure by following the branch that fits to the bit's value. Some of the nodes encountered down the trie (either branch or leaf nodes), have a prefix associated with them, so that the path from the root to them is the value of the key. These nodes can carry an associated state such as the interface value associated with the matched prefix (or key). When matching an input label while moving down the trie, the last encountered node with a matching prefix is always remembered, so that when a progress down the trie is no longer possible, the rule associated with the last matching prefix (that is also the longest) is used. An interesting aspect of the tries is that the key strings (or prefixes) do not need to be stored and can be computationally retrieved, if necessary, by taking the path from the trie root to the last encountered node with an associated prefix. However, as said above, tries are useful for matching longest prefixes in a system with hierarchical identifiers. In cases where the identifiers are flat, as in the Autonomous Systems identifiers, or present a different type of structure as in the case of our virtual coordinates, the use of tries is beneficial only from an access speed point of view and presents no advantages when it comes to compressing the routing information. Indeed, when the identifiers attributed to the network domain nodes present no hierarchical structure between them, all of the values associated with the domain labels will be stored at individual leaf nodes of the trie (i.e. one leaf node per domain). Therefore, when the associated state is a distance value, as in our use above, relying on a trie data structure would induce the same cost as the storage of the full distance vector (i.e. a distance value per network domain node).

In fact, when no hierarchy in the identifier space is assumed, one would better compare our method with those of label switching such as *MPLS* and *Ethernet* forwarding, that fall under the category of *exact label matching*. In this area of application, the dominant information

retrieval approach is *hashing*, that, when provided with a label to be matched, the label is fed to a hash function translating it into an array index. This index is then used to retrieve the information (value) associated with the label from an array storing all the values. In the particular context of Ethernet switching, hashing is highly appreciated as it is very efficient from a memory access point of view, thus allowing to perform wire-speed forwarding. Although "normal" hashing methods can have a linear complexity in terms of memory access in case of conflicts (i.e. O(N) memory access with N keys to be stored), a variant named *perfect hashing*, that adapts a parametric hashing function to the data at hand, allows to avoid conflicts and thus results in a fixed cost for the retrieval operation. However, also in the case of the hashing methods, no compression of the routing/forwarding data is performed. Indeed given that all the values associated with the labels to be hashed, be they interface identifiers or distance values, are stored in the value array, this method results in the same cost as storing the full distance vector.

Another approach to the problem of exact label matching is that of *binary search*. In the case of one dimensional label data such as AS identifiers or MAC addresses, equipped with an order primitive (be it the natural order, or lexicographic order), a binary search guarantees a label matching (retrieval) operation with a complexity of $O(\log N)$ where N is the number of labels to be stored. This is due to the organization of the data in a binary search tree data structure. The construction of such a tree proceeds by first storing the given labels in a sorted array. The median label (node stored in the middle of the array) is selected as root of the tree. As a left child of the root node, is selected the median node of the sorted array subset at the left of the root (i.e. the median of the label values inferior to the value of the root). At the right child is stored the median of the label values superior to that of the root. The construction of the tree then continues recursively by having each inner node storing the median of the values superior to it on its right branch and that of the inferior labels on its left until no further subdivision is possible. When matching a particular label value, one then is simply required to navigate down the tree following the right branch of a node if the value to be matched is superior to the encountered node's value or the left branch in the contrary case until a node with the searched value is encountered. In the case where multidimensional labels (i.e. multiple keys) are to be indexed, the extension of the binary search tree is denoted as *k-d-tree* [8]. This specific type of binary trees operates the

same way except that instead of considering the full value of the multidimensional label (that would have to be computed as a function of the different features or dimensions), the recursive splits are performed along one dimension at a time. Thus, in an example two-dimensional data, the root node would be selected as the median of all points when sorted according to the x (or first) dimension. On the next subdivision, namely the children of the root, the subdivisions are then performed along the y dimension. The construction of the tree then continues recursively in a similar way to the uni-dimensional case by looping in a round-robin fashion on the dimensions for the split. A search on this data structure also proceeds in a similar way to the uni-dimensional case by comparing the value of the multidimensional label at the dimension indicated by the node performing the split (with the value indicated by the node). The complexity of the search operation in such an algorithm is on average $O(\log N)$, however, differently from the uni-dimensional case, the worst case complexity can go up to $O(N)$ memory accesses. Although such a data structure offers a fast information retrieval on average, it shares with methods such as tries and hash tables, the disadvantage of requiring the storage of all the data points to be indexed at the interior nodes of the tree, thus performing no compression.

Other variants of the recursive space-splitting methods are *quadtrees*, *octrees* [97], and more general *R-trees* [56]. These data-structures, used mainly in graphical applications and spatial databases, allow to approximate general shapes by defining their minimal bounding boxes that are, in most cases, squares and rectangles. The quadtree, for example, initiates by subdividing the given space (assumed to be a finite two-dimensional grid) into four equal areas. In the tree data-structure, this is indicated by having four branches out of the root node. In case a space tile contains no points, or a number of data points inferior to a threshold one, no further subdivision is required. In the contrary case, the subdivision process continues recursively until it is no longer necessary. Such a scheme could be used in our case for aggregating the cloud of points received via path advertisements similar to those of figure 5.4. This would notably have the advantage over the k-d-tree method of not requiring to store all the data points as the shape of the offered area would by approximated by the tiles (of different size/granularity) indicated in the quadtree. Thus only the data defining the size and the positions of the bounding boxes (encoded in the tree) is necessary. Similarly to previously discussed methods such as convex hulls, the above methods do not guarantee a good discrimination between the set of offered and

non-offered destination coordinates. The approximated bounding boxes, around the set of offered node coordinates, may indeed contain a large number of non-offered coordinates, thus resulting in routing errors.

To summarize, when comparing our decision-tree learning approach to the set of existing methods for organizing spatial data, we find that it offers the best compromise. First due to its compression capability, as it allows to find a minimal decision surface discriminating between the data points and thus requiring to store only the tree defining that surface (thus disposing of the initial data points). Second, as it allows for fast information retrieval due to the information gain optimization. This guarantees a swift decision process, requiring only a small number of memory accesses. In addition, such a data structure presents an algorithmic simplicity, as well as a strong similarity to data structures commonly used in the networking literature. Such a similarity, makes it a viable candidate for a deployment within the networking field.

5.5 Usage for forwarding tables

Note that in most of the discussion above, the information base being indexed was mainly the routing table, or more precisely the *Adj-RIB-In* base. Our intention was to demonstrate that the received routing information from a single neighbour could be indexed and compressed, which in turn could be a motivation for maintaining it and using it for multipath routing. Indeed, as discussed above, in the actual BGP implementations, once the $Loc-RIB$ and the forwarding base computed, the information of the $Adj-RIB-In$ tables are deleted, to free resources, leaving only one path indicated per destination. This is not an optimal choice as such an information could come handy in cases of link failures or general support of multipath routing.

In case the single path model based on forwarding tables persists, our model presented above would still be applicable and would perform equally good. In a forwarding table based on virtual coordinates, an incoming packet is to be matched to an outgoing interface. Thus, instead of performing classification where the goal is to retrieve the administrative distance value to the input coordinates, we can simply choose the corresponding outgoing interface as the result of the classification process. This could reduce the size of the forwarding tables (hopefully also by 60%) and offer a fast forwarding decision process.

5.6 Extension to other network node features

As mentioned above, our use of the obtained virtual coordinates in this chapter was merely as *topological features*, fed to a feature extraction engine that is the decision tree classifier, in order to represent and access the routing data more efficiently. Therefore a legitimate question can be to ask whether other characteristics of the network nodes could be incorporated into this model, in order to allow for a more efficient routing.

A typical information about the reachability of a network node is not only its network position but also the type of services it provides. Indeed, one might be able to a access a network node at the routing level, but this does not guarantee access rights at all application levels. A typical example of this can be a network node providing only web services (HTTP protocol access) to other network nodes and rejecting any other type of access. Such an exclusion information is usually not carried along in the conventional path advertisement protocol and is therefore not visible to other nodes. For our example scenario, we imagine an extended path advertisement message (NLRI) in which a node, not only announces its reachability via its neighbours, but also the types of services it offers access to.

Thus in case the path advertisement message already contained the d dimensional virtual coordinates attributed to the node, considered as d *features* of that node, the additional type of offered service information can be considered as one additional feature describing the advertising node. Hence, our visible routing space information, analogous to that pictured in figure 5.5, is now composed of d+1 dimensional points that are also the input to our classification method. Note that in order for these points to be part of a d+1 dimensional *Real* vector space, then the d+1 dimensional feature indicating the type of offered services must be of a numeric form. One could for example imagine making use of the TCP port numbers attribution to indicate the offered services. For our example of a web server (only) node, the d+1 dimensional feature transmitted in the path advertisement message would then have a value of 80. This might be a requirement for certain types of classifiers, but is however not a necessity for our decision-tree based classification approach, as the decision tree learning algorithm can easily cope with categoric (i.e. non numeric) data. Our proposed classification method can then proceed in the same exact way described previously by trying

to find a separation surface that describes at best the multi-dimensional data.

Such a scheme offers several advantages. First it allows for a routing visibility across the conventional network layers, as a service requesting node can plan for its entire communication session just by considering information at the router level and not having to establish a communication request. Moreover, such a model can also be beneficial for the application service providers as it gives them the flexibility of indicating which types of services are to be accessed via which neighbour. A network node (or domain) offering both a web and a file transfer service can for example indicate to other network nodes that the web service is to be accessed via its neighbour A while the FTP service requests should be incoming via neighbour B. This can be easily done by having the node advertising an NLRI message containing the coordinates $< x, y, 80 >$ via A and another containing coordinates $< x, y, 21 >$ via B where (x, y) are the coordinates attributed to the node (in case of a two-dimensional embedding). In the current Internet, such an ingress influence is usually obtained by splitting the attributed IP address space into two separate prefixes and announce them as being two different ASs (domains). Our approach therefore could offer a more elegant solution.

Note that indicating such service-level details might result in an increase of the routing information. However, as shown in the evaluation results above, the classification method, requiring to store only the decision boundaries between the points representing the entries (and not the entries themselves), would hopefully allow to shrink the required data storage while still offering a fast retrieval.

The above proposed example of offered services information is just one of many possible features that could be taken into account. Other features such as congestion levels, round-trip times, trust values, and so on, could also be incorporated to the reachability message.

Chapter 6

Satisfying routing strategies

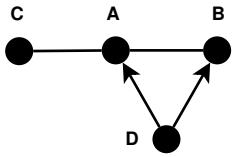


Figure 6.1: Repeated example of administrative relationship network generating multiple views on distances that was shown in figure 4.7

In this chapter, we explore the option of embedding multiple network views. or perceptions, into a single target coordinate space using a *consensus data structure* method inspired by network analysis for Social Networks. Our goal here is practically to find a common distance matrix that summarizes the different perceptions of distances. Instead of deriving the consensus distance values from the network graph, as is the case of Locally Aggregated Structures discussed in chapter 4, we define a method allowing to *search* for consensus distance values satisfying at best our objective of administrative greedy routing.

Later on we show that our distance values "customization" to the problem at hand, allows to extend the functionality of greedy routing beyond the mere point to point reachability to support features such as fault tolerance and multi-path support.

6.1 Constraints for Best-Path Routes

A possible first attempt for obtaining consensus distance values, can be based on the idea of *Locally Aggregated Structures* (LAS), a consensus approach for aggregating Cognitive Social Structures (CSS). Note that the LAS method in this case would have to be applied to distance matrices and not to the incidence matrices. To do so, one could start by gathering the different distance matrices into a single three-dimensional matrix, in which the third dimension indicates the viewer of the distances. Then, diagonal "slices" of the three-dimensional matrix are obtained by maintaining only the rows (or columns) of the nodes represented in the values as in Section 4.4.1. For the network of figure 6.1 (same as 4.7), matrix 6.1 shows the obtained consensus structure when applying a row-LAS like operation on different perceptions of network distances. These perceptions are correspondent to the matrices of page 90.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & \infty & 1 \\ 1 & \infty & 0 & 2 \\ 1 & 1 & 2 & 0 \end{pmatrix} \end{array} \tag{6.1}$$

One notices here several deficiencies with the resulting matrix values. A first problem is the incoherence of the distances with regard to the triangular inequality. The triangular inequality is a necessary condition for an embedding in a metric space. For example, taking the distances d(i,j) between nodes A, B and C in the resulting matrix (where the row index i indicates the source, and the column index j the destination), we get

$$d(B, C) > d(B, A) + d(A, C)$$

Such a violation of the triangular inequality makes an isometric embedding in a metric space, such as a low-dimensional Euclidean space, infeasible. Second, an important problem with this consensus structure is that even if a greedy embedding were admissible (e.g. in a non-metric space), the reported distances would still cause routing deadlocks. For example consider the row-vector of node A, where a distance of 1 is reported from node C. This allows practically node B to route packets destined to C through A while in reality A will refuse to relay theses packets any further. In fact such a case between the nodes A, B and C is a good example of the requirements on the consensus values of the distances. Thus, in order for node C, not to be fooled into routing greedily to B via A, the consensus distances must satisfy the following condition :

$$\boldsymbol{D'}(C, B) \leq \boldsymbol{D'}(A, B)$$

where $\boldsymbol{D'}$ is the consensus distance matrix, making C appear closer to B than A and therefore discouraging B from considering A as a greedy next hop towards C.

Hence, the order relationships of the consensus distances are actually more important than the exact distance values, when we want to comply with the loop avoidance rule of greedy routing: *"a node may forward a packet to a neighbour only if the neighbour is closer to the destination than the current node"*. Continuing in the same line of thinking, we

must also ensure that B cannot reach destination C via A, i.e.

$$\boldsymbol{D}'(B,C) \leq \boldsymbol{D}'(A,C) \tag{6.2}$$

and also that we encourage administrative authorised paths, namely C is encouraged to route via A to D,

$$\boldsymbol{D}'(A,D) < \boldsymbol{D}'(C,D)$$

that D is discouraged from forwarding packets to C through B,

$$\boldsymbol{D}'(D,C) \leq \boldsymbol{D}'(B,C) \tag{6.3}$$

or that even if B appears as a viable candidate, A is still preferred to B,

$$\boldsymbol{D}'(A,C) < \boldsymbol{D}'(B,C)$$

and that D will prefer to reach C via A

$$\boldsymbol{D}'(A,C) < \boldsymbol{D}'(D,C) \tag{6.4}$$

When combining the constraints (6.2),(6.3) and (6.4) above, we have

$$\boldsymbol{D}'(B,C) \leq \boldsymbol{D}'(A,C) < \boldsymbol{D}'(D,C) \leq \boldsymbol{D}'(B,C) \tag{6.5}$$

The conclusion is that finding a consensus distance matrix whose values admit to greedy routing (for all possible paths), is infeasible when there are conflicting constraints, as in this example above. Also, it can be shown that this is independent of the target embedding space, as no space, be it pseudo-metric, semi-metric and other variants, would allow a distance value to be strictly inferior to itself as in (6.5). Unfortunately this example is not a special case, as one may easily produce other analogous network graphs that exhibit this infeasibility. This is typically due to the conflict of interests between the network participants making a single embedding, satisfying all participants' requirements, impossible.

Nevertheless, in the remaining of this chapter we will seek a consensus data structure that minimizes the amount of such conflicting constraints and, following this, the number of greedy routing violations. The underlying hope is that, if the number of violations is small, they maybe compensated by a small amount of routing state that will lead to successful routing. Before explaining how to perform such a minimization, we first provide some background on the constraints definition mechanism.

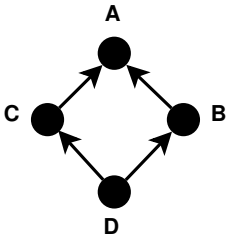
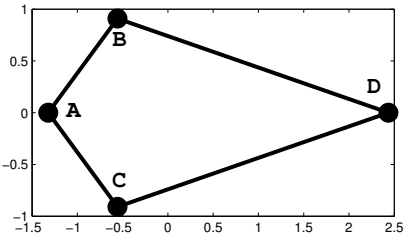


Figure 6.2: Example topology with possibility of multipath support



$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 4 \\ 1 & 0 & 2 & 3 \\ 1 & 2 & 0 & 3 \\ 4 & 3 & 3 & 0 \end{pmatrix} \end{array}$$

Figure 6.3: Multipath supporting consensus matrix of example network in figure 6.2 and its embedding

6.2 Constraints for Multi-path Support

In the example of the previous section we looked at the constraint set of the consensus distances that would need to be satisfied for administrative single path greedy routing between every source-destination pair. Another more interesting possibility could be to search for consensus distance values that allow the existence of multiple possible paths to a destination and/or even incorporate paths preferences, as we will discuss in the following section.

Consider the example of the network in figure 6.2 where node D is a client of both nodes B and C, which in turn, are clients of node A. Let us consider some of the constraints involved when representing *all possible paths* in the consensus distances for this case.

In order to force D to consider the path $D \rightarrow B \rightarrow A \rightarrow C$ in a greedy decision (in addition to the direct link), consensus distances

must be chosen such that,

$$\boldsymbol{D}'(B,C) < \boldsymbol{D}'(D,C)$$

Given the administrative relationships in place, the only other case where multiple paths are available is from node D to B through the path $D \to C \to A \to B$. This requires that,

$$\boldsymbol{D}'(C,B) < \boldsymbol{D}'(D,B)$$

Figure 6.3 shows a distance matrix whose values satisfy both of these two constraints (in addition to forbidding downhill routing through node D), and one respective feasible two-dimensional ordinal embedding of the consensus distances.

It is worth pointing that the presence of administrative policies when we are considering multiple paths, is often beneficial when searching for a distance matrix that admits greedy routes! In absence of administrative policies the number of multi-path constraints would be substantially larger and, therefore the probability of conflicts emerging, higher. For instance, when ignoring the administrative relationships in our example case, we would have to consider constraints for the additional (multiple) routes between B and C (through the valley of D),

$$\begin{cases} \boldsymbol{D}'(D,C) < \boldsymbol{D}'(B,C) \ (\textit{to take path } B \to D \to C) \\ \boldsymbol{D}'(D,B) < \boldsymbol{D}'(C,B) \ (\textit{to take path } C \to D \to B) \end{cases}$$

In this case half of the multipath constraints become unsatisfiable, and a multipath greedy embedding solution improbable. Note however, that despite this added benefit enabled by the administrative policies, one can still not guarantee the conflict free nature of the multipath constraint set, as shown in the previous example.

6.2.1 Representing Path Preferences

When considering multiple possible paths towards a destination, a natural question arises about the decision on the best or administratively most preferred route. The question is if such information must be encoded into the consensus distances and if so, how ?

Starting from the latter part of the question, an intuitive step to encode preferences is by expressing them as additional constraints. For example in the graph of figure 6.2, to force the greedy routing process

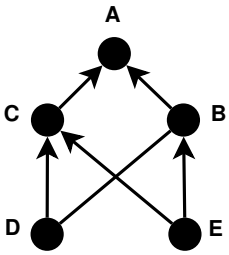


Figure 6.4: In this case both nodes D and E have multiple paths to A. Contradictions in their preferences can therefore appear

into favouring the path $D \to B \to A$ over $D \to C \to A$, one would add a constraint

$$\boldsymbol{D}'(B, A) < \boldsymbol{D}'(C, A)$$

The path preferences can be pre-configured administratively or simply follow a cost minimization logic. For example, the typical path preference rules from BGP may be adopted, such that paths via *customer* neighbours are preferred over paths via *peer* neighbours, which in turn are favoured over paths via *provider* neighbours.

The problem we see with this approach is that each node participating to the network will be imposing its own path-preference constraints on the consensus data and hence on all other nodes. Therefore the probability of contradictory constraints due to conflicting preferences increases. Let us illustrate this with another example in figure 6.4. Node E has two possible paths towards destination A via either provider B or C. If E's favourite path to A is $E \to B \to A$ and that instead, D's *only* path to A is $D \to C \to A$ (given that it has a peering relationship with B, it cannot transit to A), then both nodes E and D will be imposing on the consensus distances the following contradictory constraints, which can allow only one them to be satisfied

$$\begin{cases} \boldsymbol{D}'(B, A) < \boldsymbol{D}'(C, A) & (\textit{introduced by } E) \\ \boldsymbol{D}'(C, A) < \boldsymbol{D}'(B, A) & (\textit{introduced by } D) \end{cases} \tag{6.6}$$

Note that the consequences of constraint violations do not affect equally both nodes. If the preference constraint of node E is not satisfied, greedy routing still succeeds through a less preferred path. However, if the preference constraint of node D is not satisfied, greedy routing fails to reach node A.

This situation leads us to think that path preferences should not be "encoded" in the consensus distances, since this would increase the complexity of the constraints system. Instead, in order to decide on the most preferred path to a destination, we can simply modify the greedy routing process so as to take into consideration the preferences of the node locally, or to deduce them from the administrative relationship between the node and its neighbour. Algorithm 1, shows the proposed modification of the greedy routing process.

The *getPreference(j)* function returns the local preference for neighbour relay. In case it has been indicated by the administrator, the function returns the respective value as a preference. Otherwise the returned preference value depends on the administrative relationship to the neighbour, such that paths via customers are favoured over paths via peers and then over paths via providers.

The new greedy routing process is implemented by *forwardPacket*. Notice that the distance comparison in this case is used only to infer the reachability of a destination through a neighbour: if the neighbour is closer to the destination than the current node, this indicates that a path to the destination exists. The actual selection of the best candidate next hop is then based only on the preference heuristic (as opposed to the distance).

Another interesting function call in this algorithm is *CrossingAuthorized()* in line 17. This function checks, based on the administrative relationships with the previous node (node l that relayed the packet to the local node), and the candidate next hop, if the valley-free transit conditions are respected (see section 4.2.3). Not only this helps to quickly eliminate violating candidates as next hops (thus avoiding to compute several distances), but also reduces greedy routing failures, as the embedding may have placed neighbour nodes that do not offer transit towards a destination closer to the destination than feasible neighbours.

In summary, keeping the neighbour preferences locally to nodes (instead of integrating them into the consensus) and relying on the modified greedy routing process that takes preferences into account, limits the number of constraints on the consensus distances and increases the probability of finding consensus distance values that satisfy multipath greedy routing.

Algorithm 1 Modified Greedy Routing to account for Path Preferences

```
1: procedure GETPREFERENCE(j) ▷ returns preference for neighbour j
2:     if Preference p is predefined then
3:         return p
4:     else if j is a provider then
5:         return 0
6:     else if j is a peer then
7:         return 1
8:     else if j is a customer then
9:         return 2
10:    end if
11: end procedure
12: procedure FORWARDPACKET(dest)        ▷ main forwarding routine
13:    l ← previous node      ▷ packet was relayed to current node by l
14:    currentDist ← Distance(currentNode, dest)
15:    bestNode ← NULL
16:    for j in neighbours do
17:        if CrossingAuthorized(l,j) then
18:            if Distance(j, dest) < currentDist then
19:                if getPreference(j) > getPreference(bestNode) then
20:                    bestNode ← j
21:                end if
22:            end if
23:        end if
24:    end for
25:    if bestNode ≠ NULL then
26:        Forward packet to bestNode
27:    else
28:        Declare routing dead-end
29:    end if
30: end procedure
```

6.3 Multi-path Constraints Extraction

Algorithm 2 Constraint Extraction algorithm for multipath support

```
1: procedure ExtractConstraints
2:     for i ∈ V do          ▷ V is the set of vertices, i is the source node
3:         for j ∈ V & j ∉ neighbours(i) do▷ j is the destination node
4:             for k ∈ neighbours(i) do
5:                 if offersAccess(k, j, i) then
6:                     add a constraint: D'(k, j) < D'(i, j)
7:                 else
8:                     add a constraint: D'(i, j) ≤ D'(k, j)
9:                 end if
10:            end for
11:        end for
12:    end for
13: end procedure
```

We are now ready to discuss the process of generating the concensus distances for representing the multiple paths for the greedy routing process, from the administrative policy graph. The first step is to create the set of constraints that need to be satisfied by the distances.

Algorithm 2 is responsible for this step, through a simple heuristic. If a neighbour node offers access to a destination node a constraint forces it to appear closer to the destination than the current node. Otherwise, the neighbour is eliminated as a candidate for greedy routing by introducing a constraint that places it further from the destination than the current node. The fact that a neighbour offers access to a destination can be infered using the function *offersAccessTo()* that itself relies on received path advertisements.

This process generates a large number of constraints. In fact, each node in the network will be generating d constraints for every possible destination (that is not a direct neighbour), where d is the node's degree. Hence the total number of constraints N_c in a network with n nodes is

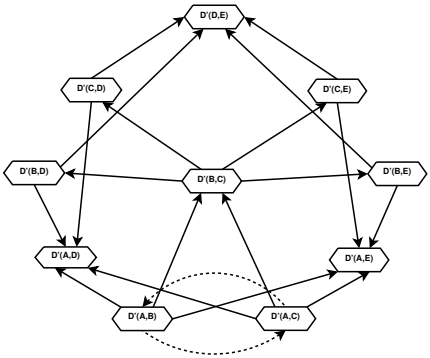


Figure 6.5: Hasse diagram representing the order constraints on the network distances in the example of figure 6.4. The dotted edges are the constraints that would be introduced due to route preferences as in equation (6.6). Such contradictions manifest as loops in the Hasse diagram

equal to :

$$N_c = \sum_{i=1}^{n} d_i * (n - 1 - d_i) = (n-1) * \sum_{i=1}^{n} d_i - \sum_{i=1}^{n} d_i^2$$
$$= 2(n-1)|E| - \sum_{i=1}^{n} d_i^2$$

where $|E|$ is the number of edges in the network. In the following section we discuss how to resolve such a large constraint system.

6.4 Solving the system of constraints

Solving the constraint system involves two steps. The first is to identify and resolve possible conflicts among the constraints. The second step then is to generate a consensus set of distance values that satisfies the constraint set. Resolving the detected contradictions practically implies the removal of the minimum number of constraints such that the remaining ones can be satisfied by a consensus solution. The hope is that the removed contradictions represent just a small minority of the constraints at hand.

The problem at hand in its general form is a central theme in Linear Programming as well as Order Theory. In Linear programming it is

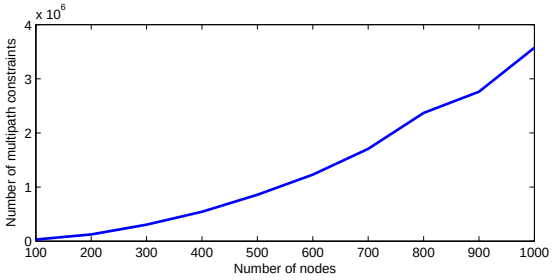


Figure 6.6: Number of generated multipath distance constraints for the test networks

addressed as an optimisation problem against an objective function by using the Simplex Method or the Barrier Method. In absence of an explicit objective function and given the fact that our constraints are actually a set of inequalities that define an order over a set of distances, we resort to a solution from Order theory[99].

We start by representing node distances as vertices in a directed graph and the inequality constraints as directed edges "heading" from vertex A towards vertex B if the inequality $A < B$ is true. This is called a *Hasse diagram*. Figure 6.5 shows such a diagram for the constraint system of the example in figure 6.4.

Any two contradicting constraints (either directly or indirectly), form loops in the Hasse diagram such as illustrated by the dashed edges in figure 6.5. Not only such a representation helps to visualize the complexity of the constraint system, but it also offers an approach to find and resolve the contradictions it contains. Specifically, resolving the constraints contradictions reduces to detecting (and breaking) loops in the Hasse diagram, which for a general directed graph can be achieved using Tarjan's algorithm [107]. This algorithm performs a variation of a *Depth-First Search* traversal to detect *strongly connected components* (SCC), i.e. subsets of graph nodes for which a path exists from each member of the SCC to all others (and hence the equivalence with loops). Once the SCCs are detected, loops can then be broken by removing one of the looping edges, i.e. one of the constraints in the system. Although such a removal would introduce routing errors (or multipath representation errors), all of the remaining edges of the SCC can still be satisfied by the consensus distances.

Once the constraints contradictions have been eliminated, the next

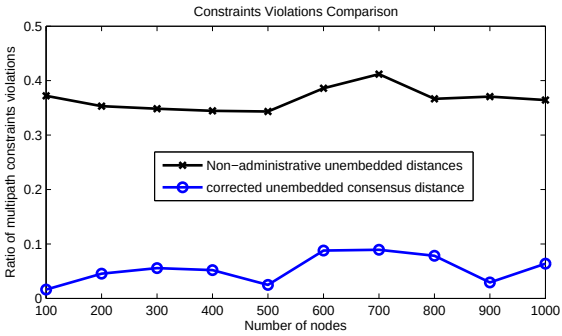


Figure 6.7: Multipath constraints violations in the case of the non-embedded non-administrative graph distances compared with the amount of infeasible constraints. Plenty of potential for improvment!

step is to generate the consensus distances.

Note that the Hasse diagram describing the constraint system has now turned into a Directed Acyclic Graph (DAG). A possible way to generate consensus distances that satisfy the greedy routing constraints can be to simply traverse the DAG starting from "root nodes" (i.e. nodes with no incoming edge, such as $\boldsymbol{D'}(A,B)$ and $\boldsymbol{D'}(A,C)$ in figure 6.5) and assigning monotonically increasing values to each traversed vertex. With such a method one can generate an infinite number of possible solutions that satisfy the non-conflicting constraints.

Note that more realistic distances can be produced by starting with the shortest distances of the network without administrative relationships and "correcting" them to satisfy the set of constraints represented by the DAG. Such a task can be achieved by an optimization method known as *isotonic regression* [29]. Given a vector of values and a set of linear constraints, the role of the isotonic regression is to find a sufficiently close, to the original vector that satisfies the given constraints.

Although the isotonic regression method is perfectly fit to our problem, the large amount of constraints on the distance values generated by our consensus approach prohibits the use of such a method in a practical manner. Instead, we have developed a heuristic distance-correction algorithm to approximate the outcome of the isotonic regression method, by using the monotonic Hasse diagram traversal explained above. First, we assign to the vertices of the Hasse diagram, the corresponding shortest path distance value in the non-administrative case. Then, starting

from the root nodes, we traverse the DAG graph updating the values of a vertex with a small threshold if it is inferior to that associated to the node preceding it. When the traversal is finished, we obtain a set of distances that violate only the infeasible set of constraints, and that are fairly close to the original distances of the non-administrative graph.

To test the feasibility of our consensus method and see if the amount of infeasible constraints (that we remove) remains reasonably small, we applied the algorithms discussed so far, on a small set of generated network graphs of sizes varying from 100 to 1000 nodes[1] and subject to administrative relationships. The small network sizes turn out to be sufficient for exhibiting the benefits of our method and at the same time it is suitable for comparison later-on, with the computationally demanding ordinal embedding technique.

Figure 6.6 shows the number of multipath constraints generated by Algorithm 2 as a function of the network size. Figure 6.7 shows the number of multipath constraints violations of the corrected consensus distances when compared with the shortest path distances on the undirected graph (i.e. when not considering administrative policy relationships). Note that this choice of comparison is due to the fact that the shortest distances in the non-administrative case, are a "natural" consensus distance structure that can be easily extracted from the network graph. The other alternative would be to compare with the LAS derived structures that present the deficiencies discussed in section 6.1. Hence, in order to evaluate the benefit of our consensus distances search approach, we will, in the rest of this chapter, compare with the performances of the non-administrative shortest distances as an alternative consensus structure.

The number of violations in our method is proportional to the number of removed constraints. As can be seen from figure 6.7, this number remains fairly low, varying around the 5% of the constraint set (and by contrast to the around 37% of the undirected graph embedding). This result suggests that a fair amount of multipath routes as well as a routing resilience is possible when routing greedily on our consensus distances.

[1] the networks were generated so as to reflect the properties of the CAIDA AS graph in terms of connectivity and administrative relationships distribution. We thank Dr. Xenofontas Dimitropoulos for providing us with this dataset

6.5 Embedding the consensus distances

Now that we have the consensus distance matrix which admits to greedy routing, the next and final step is to embed the distances in a low dimensional space, such that greedy routing can be performed using the obtained node coordinates. It turns out, as we will see in this section, that this is the most challenging task.

6.5.1 Metric embedding of the consensus distances

When embedding the obtained consensus matrix using techniques of *Metric Multi-Dimensional Scaling* (M-MDS), we get a very poor compliance to the multi-path constraints, with a violation percentage that is around 37%, far above the 5% of the non-embedded distances, as can be seen in figure 6.8 (the consensus distances are labelled as "corrected" in the figure). In fact, we observe that the resulting embedding distortions are almost as severe as if we have directly produced an embedding of the undirected graph (not accounting for the administrative policy relationships), which in figure 6.8 violates 35% of the constraint set (line labelled "*undirected metric*").

This could be due to the fact that in the process of generating the consensus distances we did not concern ourselves with a metric space where these distances may be manifested. In other words, although we made sure that greedy routing is possible on those distances, we nevertheless did not additionally cater for compliance with the triangular inequality in face of an imminent embedding to a metric space.

To correct this shortcoming, an intuitive approach is to increase the value of the consensus distances by a constant value until the triangular inequality rule is respected, and while maintaining their relative order (which guarantees their compliance to the constraint set). Estimating such a value is the goal of *Additive Constant* methods [12, 18, 95] that may produce an additive transformation for our consensus distance matrix, such that the between-node distances are guaranteed to be Euclidean in some (high-dimensional) vector space. Note that such a transformation, being additive does not modify the order of the distances and therefore preserves the constraints for the greedy routing. The only (and rather important) problem is however, the high dimensionality of the space in which the distances are Euclidean. This dimensionality can go up to $N - 1$, with N being the number of nodes in the graph!

A test with Cailliez's [18] method for estimating the additive constraints, is rather discouraging as to the effectiveness of this approach in our problem. We see in figure 6.8 (line labelled "*euclidized metric*") that the resulting constraint violations when embedding in a 2-D space, is on average around 37%.

6.5.2 Ordinal embedding of the consensus distances

What we have seen so far is that the embedding of the consensus distances in a metric space is a rather challenging task. A high-dimensional embedding can be reasonably accurate but impractical, and a low-dimensional embedding introduces distortions that lead to additional violations of constraints beyond those that we have excluded in the conflict resolution process. One strategy is to try and reduce (ideally eliminate) the number of constraint violations induced by the embedding distortions, by adapting the embedding values to respect the violated constraints. After all, as mentioned previously, in the case of greedy routing and our multipath constraints, it is the order between the distance values that matters more than the values themselves. Therefore, we are interested in an *incremental* embedding process that tries to preserve, in every step, the order of the consensus distances (i.e. guarantee that most of the multipath constraints are satisfied in the resulting embedding). This is exactly the scope of the *Ordinal Multi-Dimensional Scaling* (O-MDS) methods. Such a method preserves the desired order of the distance values in the input matrix by performing an iterative optimization task similar to the *Expectation Maximization (EM) algorithm* [32]. This process involves and iterative sequence of alternating M-MDS and isotonic regression.

First, an embedding of the consensus distance matrix is produced. In this initial embedding the set of distances D_e between the embedded points is computed. Then, isotonic regression is used for "correcting" the distance matrix D_e such that the values of D_e obey the same order as the respective in the input matrix. The resulting corrected D_e matrix is then the input of the next iteration of embedding using metric MDS. This process repeats until a termination criterion is met. Note that if the set of constraints is too large for isotonic regression (typically in our case), the heuristic correction method that we described in the previous section can be used instead.

To test and compare the effectiveness of this strategy, we applied it to the consensus distance matrix, the transformed consensus matrix

(with the additive constant for admissible Euclidean embedding), and also to the shortest path distance matrix of the undirected network graph (where administrative policies are not taken into account).

As we see in figure 6.8, the O-MDS embedding of both the consensus distance matrix and the transformed consensus matrix (labelled as "*corrected ordinal*" and "*euclidized ordinal*" respectively) achieves an improvement over the respective M-MDS embeddings (labelled as "*corrected metric*" and "*euclidized metric*" respectively). The percentage of constraint violations has dropped to about 30% (except for the example case of 700 nodes). Moreover, the performance of the consensus distance matrix seems to be slightly superior than the transformed consensus matrix. Finally, for the the shortest path distance matrix of the non-administrative network graph (labelled as "*undirected ordinal*"), O-MDS embedding provides no improvement whatsoever over the M-MDS embedding (labelled as "*undirected-metric*"), with regard to constraint violations. Hence, as expected, ordinal embedding methods allow to obtain slightly better results than the M-MDS case. The surprising fact however remains that the ordinal embedding of the "Euclidized" distances, performs slightly worse than the ordinal embedding of the consensus distances. This might be due to the increase in the inherent dimensionality of the data points when transformed to satisfy the triangular inequality condition. Indeed as previously discussed, such a transform requires a very high-dimensional embedding in order to correctly represent the transformed distances and might therefore be an explanation for the poor performances exhibited with the low-dimensional embeddings reported above.

6.5.3 Hyperbolic Embedding of the consensus distances

So far we have been attempting to find embeddings for the consensus distances in a low-dimensional (2-D) Euclidean metric space. As it has been shown in recent literature [83], often the nature of an undirected graph renders such a low-dimensional embedding difficult (or even infeasible), with regard to admissibility for greedy routing. For this reason we have also tried to embed the consensus distances in a hyperbolic metric space, where latest findings [64] show that an admissible embedding to greedy routing of an undirected graph is always possible as a greedy embedding of its spanning tree is guaranteed.

Given that the example network graphs used in our tests, are re-

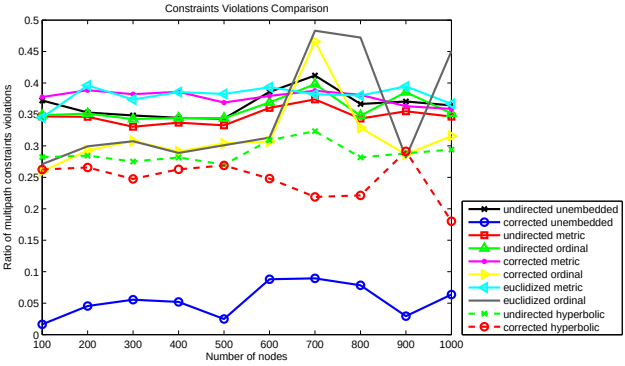


Figure 6.8: Constraints violations according to embedding method

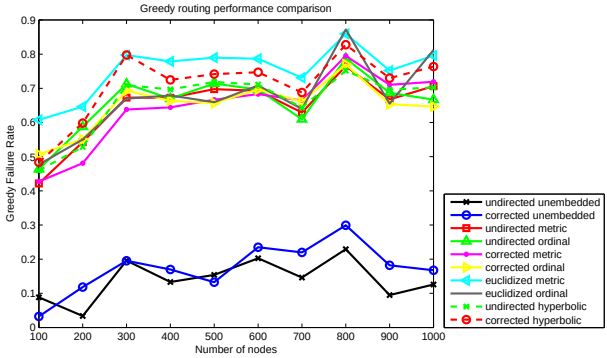


Figure 6.9: Greedy Routing failures according to embedding method

producing the Internet's Autonomous System graph properties, they essentially share similarities to a tree graphs as they belong to the family of scale-free graphs. Our intention in the following is therefore to evaluate the performances of the consensus distances when embedded in such a space, as it would be a more appropriate one for the example networks at hand. In a hyperbolic embedding, the data points are mapped to coordinates in the Poincarré Disk model, where the distance between data points exponentially increases the closer we get to the border of the disk. In order to obtain the Poincarré disk coordinates, we used

the implementation provided by Crovella et al. [26]. In the results presented in figures 6.9 and 6.8, we report the performance achieved by a hyperbolic embedding for both the shortest path distance matrix of the undirected network graph (where administrative policies are not taken into account) and the consensus distance matrix (labelled as "corrected hyperbolic").

As predicted, the use of the hyperbolic coordinates seems to improve the performance with regard to the multipath constraints violations, achieving a violation percentage (for the consensus distances) below 20%. Nevertheless, when looking at the greedy routing success rate, using our modified greedy algorithm (that takes into account administrative policies) the hyperbolic embedding does not perform any better than the other embedding techniques considered so far, with a failure rate above the 70%.

6.5.4 Summary on embedding methods

As shown by the results above, the embedding procedure, independently of its specifics (Euclidean, hyperbolic, ordinal etc.), fails to exploit the full potential of the computed multipath capable consensus distances. When compared with the multipath constraints violations prior to the embedding, all the embedded consensus distances (according to different embedding methods) violate much more constraints and exhibit results similar to those of the undirected graph distances (i.e. shortest distances without considering administrative relationships). Moreover, when considering the greedy failure rate performances shown in figure 6.9, one can see that none of the embedding procedures, when applied to both non-administrative graph distances as well as the *corrected* ones (i.e. our consensus distances computed above) provide useful results. Indeed, as shown in figure 6.9, all of the embedding methods seem to converge with the increase in the network size to a greedy failure rate (when subjected to administrative constraints) at around 70% thus making a practical usage of the consensus distances infeasible.

6.6 Transform coding of the consensus distances

Given the discouraging results attained so far by trying to embed the consensus distances in a low-dimensional metric space, we attempt a

change of strategy in this section onwards. Initially, our motivation for attempting a metric-space embedding was to enable (almost) stateless greedy routing on the embedded node coordinates. This practically meant

(i) Near-zero storage for routing state
(ii) Fast forwarding decisions with almost zero memory accesses

In face of the dead-end in finding a sufficiently well-performing solution we decide to relax our initial requirements, and seek for a solution that meets the following goals

(i) Sufficiently small storage required for routing state
(ii) Fast forwarding decision requiring only a few memory accesses

Therefore, we search for alternative ways of exploiting the consensus distance matrix and its multipath routing support information.

Starting with the redefined first goal, when provided with the full distance matrix data, we start considering compression methods to reduce the amount of data to be stored. Such methods would either enable us to store the exact distance between two network nodes in case of lossless compression, or an approximate distance in case of lossy compression, the quality of which depends on the compression ratio.

Lossless compression methods have two main variants, the *Huffmann coding* and *Arithmetic coding*. Both methods build a statistical model of the data by measuring the probabilities (frequencies) of all symbols (values) being present in the dataset. From this they construct an encoding scheme requiring only a few bits for the most frequent symbols (instead of an equal number of bits for all symbols) and thus result in reduced storage for the presented dataset. Although we can envision similar methods applied on our consensus distance matrix, they would however still require the same number of records in the routing table (as the distance matrix), albeit with a less storage required per record.

Another family of compression methods are the lossy and more particularly *transform coding methods*. These are commonly used for image compression (an image is by nature a 2-D matrix!), by retrieving and deleting the most irrelevant details in the dataset, and maintaining only values carrying most of the information. The most prominent techniques of transform coding for digital signals represented as vectors and matrices, are the *Discrete Fourier Transform*, the *Discrete Cosine Transform*, the *Karhunen-Loève Transform* and the *Wavelet Transform*.

The Karhunen-Loève Transform (KLT), also know as *Principal Component Analysis* (PCA), represents the values in a matrix as a linear combination of its eigenvectors and eigenvalues. In the more particular case of symmetric matrices, that is also our case, this transform is in fact equivalent to a spectral spectral decomposition of the matrix, computable via an eigendecomposition. This decomposition allows to find an orthonormal matrix $\boldsymbol{Q}$ (i.e $\boldsymbol{QQ^T} = \boldsymbol{I}$) and a diagonal matrix $\boldsymbol{\Lambda}$, such that

$$\boldsymbol{D} = \boldsymbol{Q\Lambda Q^T}$$

where the columns of the $\boldsymbol{Q}$ matrix are the eigenvectors of the matrix $\boldsymbol{D}$ and the diagonal values of $\boldsymbol{\Lambda}$ are the corresponding eigenvalues. When considering this decomposition as follows

$$\boldsymbol{D} = (\boldsymbol{Q\Lambda})\boldsymbol{Q^T} = \lambda_1 \boldsymbol{q_1 q_1^T} + \lambda_2 \boldsymbol{q_2 q_2^T} + ... + \lambda_n \boldsymbol{q_N q_N^T}$$

we obtain a *spectral decomposition* of the matrix $\boldsymbol{D}$ into a sum of matrices, as each *outer product* $\boldsymbol{q_i q_i^T}$ produces a matrix of the same size as $\boldsymbol{D}$. Note that the "contribution" of these component matrices to the matrix $\boldsymbol{D}$ is proportional to the magnitude of the corresponding eigenvalue λ_i, meaning that the matrices associated with a large eigenvalues carry most of the information and those with smaller values indicate less relevant details. Hence, compressing our distance matrix using the KLT transform can be achieved by storing only $k < N - 1$ of the $\boldsymbol{q_i}$ eigenvectors in a decreasing order of their eigenvalues magnitude. The advantage of the KLT transform over other transform coding methods is that it is the most adapted to the data at hand in a least squares sense [85] (by producing a transform *basis* for each dataset).

By contrast, other transform coding methods such as the Cosine and the Wavelets transform for example, use a fixed basis on all datasets to be compressed. A fixed basis has the advantage of not requiring to transmit the transform basis along with the compressed data. For example, in the case of the KLT transform a compressed image would be impossible to decode unless the eigenvectors forming the basis of the transform are also communicated with the image data. Fixed basis transforms include the Discrete Fourier Transform (where the basis are the sine and cosine functions of different frequencies), the Discrete Cosine transform (the basis is the cosine function at different frequencies) and the Wavelet Transform (the basis is formed by a set of orthogonal functions called wavelets).

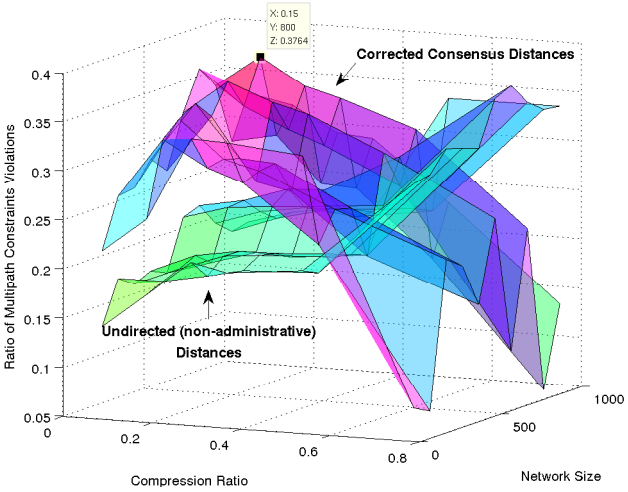


Figure 6.10: Multipath constraints violations when using the KLT transform on both the consensus distances and the undirected distances

6.6.1 Using the KLT transform

Given that the KLT (PCA, SVD) transform provides us with the basis that is best fit to our distance data at hand, and considering the fact that the compression is meant to be node-local (i.e. no transfer of compressed data between nodes), we chose to apply it to the computed consensus distances and see if better performances than the embedding methods can be achieved. In order to measure the benefit of our consensus distance matrix computation through the constraints resolution process, we also applied the KLT transform method to the undirected distance matrix, i.e. the shortest path distances between network nodes without considering any administrative concerns. The obtained results for the multipath constraints violations and the greedy routing failures are respectively shown in figures 6.10 and 6.11. A general comment about both results is that better performances are achieved when compared to the embedding strategies. A surprising result however is the clear dominance at low compression ratios of the KLT transform when using the undirected distance matrix over our computed consensus distance matrix. Indeed, when considering the greedy failure rate results

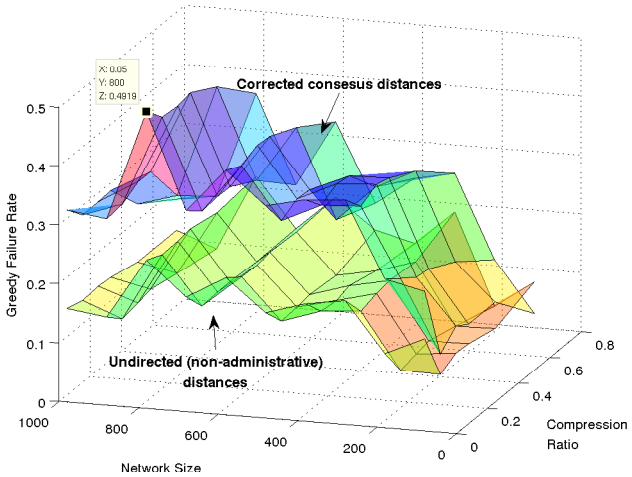
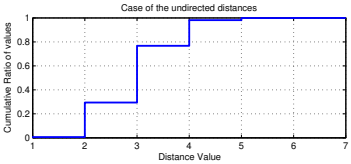
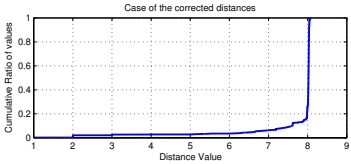


Figure 6.11: Greedy Failure Rates with administrative concerns when using the KLT transform on both the consensus distances and the undirected distances



(a) Cumulative distribution of the undirected distance values



(b) Cumulative distribution of the corrected distance values

Figure 6.12: Difference of the distance value distribution between the undirected and the corrected distances for the example case of 600 nodes

on figure 6.11, one can see that the KLT transform applied on the consensus distance matrix performs worse than when applied on the undirected distances and this independently from the network size and the desired compression ratio. An even more interesting result is visible in figure 6.10. In this case, when the desired compression ratio is high (75% for example), meaning that both initial matrices are rather well

approximated, the performances of the approximated distances with regard to multipath constraints violations are similar to those of the initial matrices visible in figure 6.7 where the corrected consensus distances clearly outperform the undirected distance matrix. However, as the compression ratio decreases, the performances achieved by our consensus distances tend to worsen which is to be expected as the matrix approximation is less and less precise, thus more likely to violate the order constraints on the distances. What is however surprising, is the effect of the compression ratio reduction on the KLT transform of the undirected distance matrices as it seems to have the reverse effect, namely that of enhancing the performances in terms of constraints violations. Indeed, the KLT transform on the undirected distances in figure 6.10 achieves the best results in terms of multipath constraints violations at the lowest compression ratio of 5% with results slightly above the 15% of constraints violations, thus getting close to the (boundary) performances of the original corrected distance matrix. Such an effect could perhaps be explained by the differences between the distribution of values in the two matrices. When applying our constraint correction methods based on the Hasse diagram traversal, a few high distance values (with superiority constraints on them) tend to drag a large part of the distance values towards them thus creating a cleavage between two groups in which the order "steps" are of very different magnitude. An example of the undirected and corrected distance values distribution for the example case with 600 nodes is given in figure 6.12. A possible explanation for the exhibited results can therefore be that in such a context where the data points are compacted (clustered), a poor approximation of the values might have more severe consequences than in the sparser case.

6.6.2 Summary on transform coding

As can be seen from the results above, acceptable performances in terms of multipath constraints violations and greedy failure rate can be achieved at rather low compression ratios even without relying on our consensus distances. An important drawback of this solution however, affecting the second main objective of greedy routing that is fast forwarding decision is the rather large amount of coefficients involved in the computation of a distance towards a destination. Indeed, although a compression ratio of 15% can sound appealing from a memory usage aspect, it also means that $0.15 * N^2$ coefficients will be involved in

the computation of the distance of a neighbour to a destination thus potentially slowing down the forwarding decision. A transform coding based solution as the one presented above might therefore not be suited to most application scenarios. For cases where the network nodes are however willing to trade computation against memory usage, the above solution might be interesting. One such case is in fact that of Ethernet switches having to process incoming packets at wire speed. In the classical literature of network switching, the complexity of the decision operation is computed in terms of memory operations, due to the fact that a memory access is much more time consuming than computational instructions. Furthermore, special fast access memories have been developed in this context allowing to considerably reduce the information retrieval time. Therefore the reduced amount of information required when using our transform coding method could allow to entirely fit the required coefficients in a fast access memory allowing to completely avoid any external memory slow accesses.

Another application field where such a method could also be applied is *Network on Chips (NoC)* [9] where processing units in a *Very Large Scale Integration (VLSI)*, rely on datagram communication paradigms for exchanging data fast and asynchronously. In these architectures routing and switching operations replace the more classical cross-bar and bus models. Processing units on those chips share the same operational incentives and challenges with network switches in the aforementioned Ethernet switch example. Being computational by nature, these units are likely to trade computational power based on information stored on registers, etc. instead of relying on external memory accesses.

6.7 Approximate Distance Oracles on the consensus distances

Another approach to distance matrix compression was proposed by Thorup and Zwick in their *approximate distance oracles* [110] method. In this proposal, the network distances are approximated through the storage of a sub-sample of the distance values. Similar to the Lipschitz methods and those of compact routing discussed in chapter 2, the main idea is that the distance between a pair of nodes u and v can be approximated (bounded) with the distance values of both nodes u and v to a third reference point w, by using the triangular inequality condition.

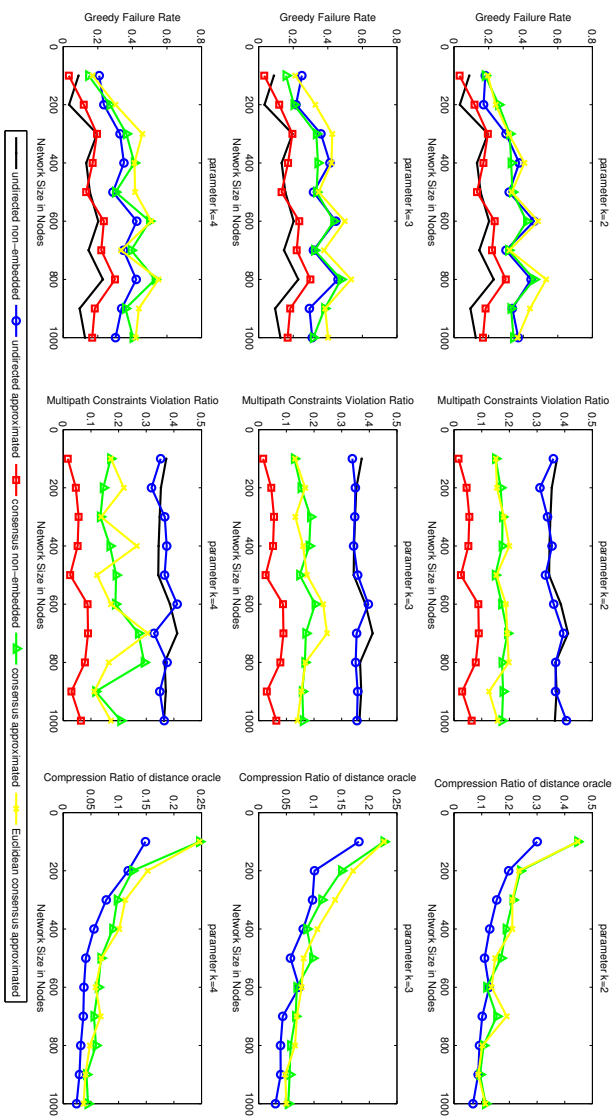


Figure 6.13: Approximate Distance oracles results comparison when varying the parameter k

Thus, a necessary condition for the efficient use of such a method is that the triangular inequality condition on the distances holds. As discussed above, the consensus distances generated through our constraints resolution mechanism do not necessarily satisfy this condition. Nevertheless, the obtained distance values can be transformed so as to fulfil this condition by simply adding a large enough constant to all distance values as discussed in [12, 94]. The principle behind such a transform is very simple. Consider all the strictly positive values c for all node triplet i, j and k violating the triangular inequality :

$$c = \boldsymbol{D'}(i,j) - \boldsymbol{D'}(i,k) - \boldsymbol{D'}(k,j) > 0$$

and consider c_{max} to be the highest such value for all the triplets (i,j,k), (i.e. the highest triangular inequality violation). Then by simply adding c_{max} to all the values of $\boldsymbol{D'}$ we obtain the matrix $\boldsymbol{D''}$ where :

$$\begin{aligned}\boldsymbol{D''}(i,k) + \boldsymbol{D''}(k,j) &= \boldsymbol{D'}(i,k) + \boldsymbol{D'}(k,j) + 2 * c_{max} \\ &= \boldsymbol{D'}(i,j) + c_{max} = \boldsymbol{D''}(i,j)\end{aligned}$$

hence having the highest triangular inequality violation corrected in $\boldsymbol{D''}$, meaning that all other smaller violations are in turn corrected. Given that the consensus distances have been transformed through a simple shift with an additive constant, the greedy routing constraints are guaranteed to remain satisfied as the inter-value order remains the same. As discussed in [94] and [18], the additive constant c_{max} can furthermore be selected to not only guarantee the metric compliance of the transformed distances but also to obtain *Euclidean* distances. Hence, by applying the constant shift operation on our consensus distances, we can make them viable to be used with the *approximate distance oracle* method.

Figure 6.13 shows the obtained results when applying the approximate distance oracles method to the consensus distances computed for the 100 to 1000 nodes example cases studied above. In this case, we approximate the distance values of the non-administrative distances (labelled as *undirected* in the figure), along with the values of the consensus distances obtained through the constraints resolution process as well as those of the transformed consensus distances satisfying Euclidean properties. Reported are the performances of greedy administrative routing, multipath constraint violations as well as the obtained compression ratio with regard to different values of a parameter k specific to the distance oracle method. This parameter influences the amount of distance

data that is required to be stored on each node, that is in the range of $O(kn^{1+1/k})$ where n is the number of nodes in the network. Note that this parameter also influences the quality of the distance approximation as the distance oracle method guarantees a stretch of factor $2k-1$, provided that the triangular inequality holds.

In order to highlight the effect of distance approximation when compared with the original distance values, we also re-plot the greedy administrative failure rate as well as the multipath constraints violations for both non-embedded non-administrative distances (labelled as *undirected non-embedded* in figure 6.13) as well as the non-embedded consensus distances that were previously reported in figure 6.7.

A first important remark when studying the multipath constraints violations results is that the performances of the approximated consensus distances as well as of the approximated Euclidean consensus distances are rather close to those of the non-embedded consensus distances, mainly remaining below the 20% violation rate. When contrasting with the non-administrative distance values and their approximation through the distance oracle methods, one can see a diminution of over 15% in the violations rate, thus meaning that the consensus distances approximation allows for a better multipath support and a fault tolerant greedy routing. When observing the variation of the greedy routing performances as a function of the parameter k, one can see that the failure rate tends to remain close to the performance of both non-embedded consensus and non-embedded non-administrative distances, with about only 10% difference in the case where $k = 2$. Note that the exhibited greedy routing performance with about 30% failure rate might seem rather high and thus unsatisfactory. One should however keep in mind that such a failure rate is measured when considering a routing procedure between all pairs of nodes. Therefore, when considering a real-world network, in which the established communications are only a small sample of all possible node pair communications (as nodes mainly communicate with a few content provider nodes), such a greedy failure rate might result in much fewer failures in practice. Also when considering the achieved compression ratio reaching the 5% for the case of 1000 nodes when $k = 2$, such a greedy failure rate can be considered rather fair compared to the enormous storage gain.

Another interesting comment here is that, although the distance oracle method highly relies on the metric nature of the provided distance values in order to offer a good approximation, this does not seem to be relevant in our application needs. Indeed when comparing the perfor-

mances of both the consensus distances and their transformed version satisfying Euclidean (and thus metric) properties, one can see that they are generally similar for both the greedy routing success as well as for the multipath constraints violations.

To summarize, by using the approximate distance oracle method on the consensus distances, we can see that a better multipath constraints support (and thus also fault tolerance) can be achieved for similar greedy administrative routing failure rates as well as similar compression ratio as in the case of transform coding methods. The major advantage of the approximate distance oracle method over transform coding ones discussed above is that the cost of the distance approximation is constant and is of complexity $O(k)$ [110], when compared with the $O(r * N^2)$ cost in the transform coding case, where r is the achieved compression ratio and N the number of network nodes. Hence relying on the approximate distance oracles method for compressing the obtained consensus distances clearly better fulfils the two goals of greedy routing usage, that are minimal routing data storage and fast forwarding process.

6.8 Discussion and future work

So far in our exploration to find an embedding solution that admits to multipath greedy routing we have not considered all the possible consensus matrices satisfying the constraints. Instead we have experimented with only one candidate constraint-complying distance matrix and several embedding and compression algorithms are applied on it. However the number of possible consensus distance matrices is infinite as they can be generated by simply traversing the DAG representing the order constraints (Hasse diagram).

An interesting follow-up exploration direction is to understand if any of the candidate solutions of possible distance matrices, is more suited for a metric embedding or the compression algorithms applied on it. Such a solution space exploration is usually achieved though optimization algorithms such as *Semi-Definite* and *Linear Programming* (so long as the constraints on the distances are linear). As a first step we would need to define a suitable objective function for assessing the quality of a distance matrix with regard to the algorithms used.

Another track we would like to pursue before condemning the embedding solutions are the Lipschitz methods [61]. Given that the obtained distances can be transformed so as to fulfil the triangular inequality

condition, the hope is that a distance approximation method à la Lipschitz may become feasible. The distance between two nodes could be approximated by their relative distances to a third landmark node. Although Lipschitz coordinates would contradict our general thrive for vector-space based (geometric) coordinates, the question is whether they could nevertheless lead to better greedy routing performance as it was the case for the approximate distance oracles.

Regarding the exploration of further ideas in transform coding in order to reduce the amount of stored coefficients, we are considering to test the use of different compression ratios at different nodes of the network. Depending on their position in the network and their communication needs, different nodes might have different needs for accuracy of the consensus distance matrices (e.g. less central nodes probably requiring a less precise approximation than core nodes).

Finally, an interesting by-product that emerged out of our approach, whose effects we would be interested to know, is the following distributed consensus requirement. In order to resolve the global constraints system, each node (domain), participating in the network is required to submit its constraints on the distance values, thus revealing its network policies to the node(s) that use this information. This is in conceptual conflict to the current approach of policy management in BGP, where a node refusing to offer relay service, simply stops propagating the received advertisements, thus keeping this refusal information local. This somewhat counter-intuitive philosophy, actually promotes cooperation incentives instead of independent stand-alone decisions for the network to function correctly.

Chapter 7

Conclusion

7.1 Summary

Our goal in this thesis was to improve the geometric greedy routing techniques, in order to make them better candidates for application in future networks. To this goal, we mainly identified two immediate challenges.

A first challenge, we addressed in chapter 3, is that of greedy routing dead-ends. In our proposed solution, we adopted a novel multi-resolution embedding of the network graph and modified the greedy routing algorithm accordingly. We were then able to obtain better routing results than high-dimensional embeddings of the network graph, with an equivalent cost in data storage. Our proposed approach of multi-resolution embedding is a novel evolution direction for the field of failure recovery in geometric greedy routing, extending beyond the classical face routing and scoped flooding.

As a second challenge, we were the first to identify the support of administrative policies as a requirement for the admittance of greedy routing techniques in both present and future networks. While studying the feasibility of this feature, we showed that, more sophisticated virtual coordinates attribution techniques than the present ones need to be considered. As discussed in chapter 4, the differentiated services introduced by the administrative relationships, induce different perceptional views of the network that are not possible to combine within a single embedding. To do so, we extended the current greedy geometric routing techniques in two fashions.

In a first approach, we used classification methods to find a distance function that better suits the perception of each node. We started by extracting the *features* of the participating nodes that consisted, in our case, in their topological position. Then, we developed a decision process differentiating distance data received through a neighbour node. The main intuition is that, if the service differentiation is dependant on the network node features, then the classifier should be able to *learn* the process causing the differentiation. By doing such a learning, we build a classifier able to reproduce the differentiation based on the input node features that are now used as virtual coordinates. This allows for a compression of the routing control data since the classifier encodes only the decision surface that requires less storage than the received feature data (virtual coordinates).

In a second approach to the problem of multiple perceptions, instead of modifying the distance *function* operating on the virtual coordinates,

we simply modified the distance *values*. The main idea was to *search* for consensus distance values that satisfy at best the perceptions of the different participating nodes, and their expectations from the greedy routing process. In fact we showed that the consensus distance values can be *forged* so as to satisfy more complex goals than basic point to point routing. By manipulating the distance values, we could extend the functionality of greedy routing to support not only administrative policies, but also multi-path and fault-tolerant routing. This method is in fact another demonstrator for the evolution capability of greedy routing. By simply modifying the distance values on which the algorithm operates, and without modifying the actual routing algorithm, we could significantly change its behaviour.

Stepping back to evaluate our contribution to the greedy geometric embedding and routing field, one can see that the two above proposals are in fact novel development alternatives. Indeed when considering the evolution of geometric greedy embeddings, one can see that the target embedding space and the corresponding distance function, were always priorly fixed, independently of the application scenario. First, the community mainly focused on low-dimensional Euclidean spaces as a target embedding space, until Papadimitriou et al. [83] showed that it is not the most ideal one. Then following the work of Kleinberg [64], the community recently converged on the Hyperbolic plane as a favourite space for embedding, and most recent works focus on methods for finding the best hyperbolic virtual coordinates, given a network graph. Our classifier proposal, in chapter 5, is therefore a considerable shift in the approach to distance-based greedy routing, where instead of priorly fixing the target space and searching for the best possible coordinates, we priorly fix the virtual coordinates and search for the best possible distance function, thus adapting it at best to our goals. This has the advantage of more easily complying with the multiple perceptions of the network, as all participating nodes are required to agree on the same virtual coordinates while remaining free to adapt the distance function to their perception.

Also, our distance values modification method proposed in chapter 6, can be seen as a pre-processing phase, prior to the embedding procedure, allowing to influence the resulting embedding (and hence embedded distance values), to satisfy a particular goal. This method, similar to the notion of *conditioner* in data analysis, would hopefully inspire other developments prior to the embedding step.

A final remark is that, all along the thesis, we heavily relied on vec-

torial data analysis tools developed within other computational fields. Our intention was to demonstrate our argument about the variety of already existing tools, made available through the use of virtual coordinates, that could be leveraged to solve networking problems.

7.2 Outlook

Many extensions of our above proposed methods remain to explore. In the case of the classification-based forwarding proposed in chapter 5, a natural extension would be to include additional node features in the decision process. Indeed, basing the discrimination process on other node characteristics than their topological position could enhance the quality, and perhaps the speed of the classification. In addition to the features discussed in section 5.6, an interesting feature in the particular case of valley-free routing on administrative relationships can be the *tier-indicator* of the node. This feature informs about the role played by the node in the global network by indicating, for example, a tier value of 1 or 2 for the main core domain nodes, and higher values for others. A the tier-indicator is in fact correlated with the the number of peers, customers and provider neighbours a node has, these values might as well be included in the classification process.

When considering our distance pre-processing method described in chapter 6, an immediate enhancement would be the distribution of the constraints resolution process. Indeed, in order to comply with the current model of administrative relationships handling, we need to find a more distributed fashion of computing the consensus distances, that does not reveal the internal policies of the nodes. One possible direction towards this goal would be node replication that consists, in making a node appear as two or more separate entities that each satisfy a given perception of the network. Note that this would be in fact equivalent to multiple address assignment.

This replication process can be achieved in two ways. A first way would be to perform replication in the network graph prior to any distance extraction and embedding. In fact such an idea is similar to the NIRA proposal [120] where a node is attributed different hierarchical identifiers according to its relationship to its neighbours. A straight forward improvement of this method would then be to enhance it beyond basic hierarchical forwarding and employ more flexible virtual node coordinates.

Node replication can also be achieved in the embedding space, by associating several multi-dimensional points with a single network node. This can be achieved through the use of the methods of *Multidimensional Unfolding (MDU)* [16] and *Correspondence Analysis (CA)*. These methods, close to the MDS family, allow to embed rectangular distance matrices by associating virtual points to the row indices, that are of a different type than those corresponding to the column indices. Thus by associating with a network node A, several distance row vectors, each satisfying a different node's perception of A's distances, we could obtain as many virtual coordinates corresponding to different perceptions of node A's role. The greedy routing process would then have to be adapted to the presence of multiple coordinates.

Another usage of our above presented methods would be to consider routing between different typed of entities such as the host-to-data model discussed in chapter 1. Here again, the *MDU* and *CA* methods, allowing to embed bipartite datasets, could be the most promising track.

Finally, we would like to consider the possible application of our above proposed methods beyond the scope of communication networks and extending to novel fields such as Networks On Chips. Although the community in this field does not yet consider greedy routing as a candidate solution, the problem requirements are in fact so similar that it could be only a matter of time. In this context, the problem of greedy routing on administrative relationships might also become relevant, as the connectivity between the inner-switches could be also subjected to (hardware) policies.

Also, when considering the current effervescence of the new field of *Network Science*, focusing on the study of network data structures – be they telecommunication, social, or other networks, we hope that our proposed methods of multi-resolution embedding, perceptional greedy traversal, and distance values pre-processing can reveal useful to other fields of study.

List of Figures

Bibliography

[1] BGP reports : http://bgp.potaroo.net/.

[2] The Cooperative Association for Internet Data Analysis : CAIDA. http://www.caida.org/home/.

[3] Ittai Abraham, Yair Bartal, and Ofer Neiman. Embedding metrics into ultrametrics and graphs into spanning trees with constant average distortion. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 502–511, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[4] Ittai Abraham and Cyril Gavoille. On approximate distance labels and routing schemes with affine stretch. In 25^{th} *International Symposium on Distributed Computing (DISC)*, volume 6950 of *Lecture Notes in Computer Science (ARCoSS)*, pages 404–415, Rome, Italie, 2011. Springer.

[5] A.Moitra and T.Leighton. Some results on greedy embeddings in metric spaces. In *Proceedings of the 49th Annual Symposium on Foundations of Computer Science (FOCS)*, 2008.

[6] Seema Bandyopadhyay and E.J. Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1713 – 1723 vol.3, march-3 april 2003.

[7] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396, 2002.

[8] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[9] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Comput. Surv.*, 38(1), June 2006.

[10] I. Blake and J. Gilchrist. Addresses for graphs. *Information Theory, IEEE Transactions on*, 19(5):683 – 688, sep 1973.

[11] Marián Boguñá, Fragkiskos Papadopoulos, and Dmitri Krioukov. Sustaining the internet with hyperbolic mapping. *Nature Communications*, 1(6), 09 2010.

[12] I. Borg and P.J.F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

[13] Prosenjit Bose, Pat Morin, Ivan Stojmenović, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 11 2001.

[14] J. Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52:46–52, 1985.

[15] Nirupama Bulusu, John Heidemann, and Deborah Estrin. GPS-less low cost outdoor localization for very small devices. *IEEE Personal Communications Magazine*, 7(5):28–34, October 2000.

[16] Frank Busing. *advances in multidimensional unfolding*. PhD thesis, Leiden University, April 2010.

[17] M. Caesar and J. Rexford. Bgp routing policies in isp networks. *Network, IEEE*, 19(6):5 – 11, nov.-dec. 2005.

[18] Francis Cailliez. The analytical solution of the additive constant problem. *Psychometrika*, 48:305–308, 1983. 10.1007/BF02294026.

[19] Qing Cao and Tarek Abdelzaher. Scalable logical coordinates framework for routing in wireless sensor networks. *ACM Trans. Sen. Netw.*, 2(4):557–593, 2006.

[20] Guanhong Pei Cedric Westphal. Scalable routing via greedy embedding. In *IEEE INFOCOM 2009 - The 28th Conference on Computer Communications*, 2009.

[21] Victor Chepoi, Feodor F. Dragan, Ilan Newman, Yuri Rabinovich, and Yann Vaxès. Constant approximation algorithms for embedding graph metrics into trees and outerplanar graphs. In *Proceedings of the 13th international conference on Approximation, and 14 the International conference on Randomization, and combinatorial optimization: algorithms and techniques*, APPROX/RANDOM'10, pages 95–109, Berlin, Heidelberg, 2010. Springer-Verlag.

[22] Fan R. K. Chung. *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92)*. American Mathematical Society, February 1997.

[23] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165 – 177, 1990.

[24] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[25] Trevor F. Cox and M.A.A. Cox. *Multidimensional Scaling, Second Edition*. Chapman and Hall/CRC, 2 edition, 2000.

[26] Andrej Cvetkovski and Mark Crovella. Multidimensional scaling in the poincaré disk, 2011.

[27] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, pages 15–26, New York, NY, USA, 2004. ACM.

[28] Jan de Leeuw. Applications of convex analysis to multidimensional scaling. In J.R. Barra, F. Brodeau, G. Romier, and B. Van Cutsem, editors, *Recent Developments in Statistics*, pages 133–146. North Holland Publishing Company, Amsterdam, 1977.

[29] Jan de Leeuw, Kurt Hornik, and Patrick Mair. Isotone optimization in r: Pool-adjacent-violators algorithm (pava) and active set methods. *Journal of Statistical Software*, 32(5):1–24, 10 2009.

[30] V. de Silva and J. Tenenbaum. Sparse multidimensional scaling using landmark points. Technical report, Stanford University, 2004.

[31] Vin De Silva and Joshua B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Advances in Neural Information Processing Systems 15*, volume 15, pages 705–712, 2003.

[32] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[33] Kedar Dhamdhere, Anupam Gupta, and Harald Räcke. Improved embeddings of graph metrics into random trees. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, SODA '06, pages 61–69, New York, NY, USA, 2006. ACM.

[34] Raghavan Dhandapani. Greedy drawings of triangulations. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 102–111, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[35] Xenofontas Dimitropoulos, M. Ángeles Serrano, and Dmitri Krioukov. On cycles in as relationships. *SIGCOMM Comput. Commun. Rev.*, 38:102–104, July 2008.

[36] L. Doherty, K.S.J. pister, and L. El Ghaoui. Convex position estimation in wireless sensor networks. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2001.

[37] David Eppstein and Michael Goodrich. Succinct greedy graph drawing in the hyperbolic plane. In Ioannis Tollis and Maurizio Patrignani, editors, *Graph Drawing*, volume 5417 of *Lecture Notes in Computer Science*, pages 14–25. Springer Berlin / Heidelberg, 2009.

[38] Christos Faloutsos and King-Ip Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *SIGMOD Rec.*, 24:163–174, May 1995.

[39] Qing Fang, Jie Gao, L.J. Guibas, V. de Silva, and Li Zhang. Glider: gradient landmark-based distributed routing for sensor networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 1, pages 339 – 350 vol. 1, march 2005.

[40] R. Flury, S.V. Pemmaraju, and R. Wattenhofer. Greedy routing with bounded stretch. In *INFOCOM 2009, IEEE*, pages 1737 –1745. EthZ, 2009.

[41] 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 sensornets. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 329–342, Berkeley, CA, USA, 2005. USENIX Association.

[42] P. Francis, S. Jamin, Cheng Jin, Yixin Jin, D. Raz, Y. Shavitt, and L. Zhang. Idmaps: a global internet host distance estimation service. *Networking, IEEE/ACM Transactions on*, 9(5):525 –540, oct 2001.

[43] Hannes Frey and Ivan Stojmenovic. On delivery guarantees of face and combined greedy-face routing in ad hoc and sensor networks. In *Proceedings of the 12th annual international conference on Mobile computing and networking*, MobiCom '06, pages 390–401, New York, NY, USA, 2006. ACM.

[44] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 21(11):1129–1164, November 1991.

[45] Hristescu Gabriela and Farach Martin. Cluster-preserving embedding of proteins. Technical report, Rutgers University, Piscataway, New Jersey, 1999.

[46] Emden R. Gansner, Yehuda Koren, and Stephen North. Graph drawing by stress majorization. In *GRAPH DRAWING*, pages 239–250. Springer, 2004.

[47] Jie Gao, Leonidas J. Guibas, John Hershberger, Li Zhang, and An Zhu. Geometric spanner for routing in mobile networks. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '01, pages 45–55, New York, NY, USA, 2001. ACM.

[48] Cyril Gavoille, Michal Katz, Nir Katz, Christophe Paul, and David Peleg. Approximate distance labeling schemes. In Friedhelm auf der Heide, editor, *Algorithms - ESA 2001*, volume 2161

of *Lecture Notes in Computer Science*, pages 476–487. Springer Berlin / Heidelberg, 2001.

[49] Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *J. Algorithms*, 53:85–112, October 2004.

[50] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[51] C. Gkantsidis, M. Mihail, and E. Zegura. Spectral analysis of internet topologies. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 1, pages 364 – 374 vol.1, march-3 april 2003.

[52] Brighten Godfrey, Kevin Fall, Gianluca Iannaccone, and Sylvia Ratnasamy. Routing tables: Is smaller really better? In *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*, 2009.

[53] R. Graham and H. Pollak. On embedding graphs in squashed cubes. In Y. Alavi, D. Lick, and A. White, editors, *Graph Theory and Applications*, volume 303 of *Lecture Notes in Mathematics*, pages 99–110. Springer Berlin / Heidelberg, 1972. 10.1007/BFb0067362.

[54] R. L. Graham and H. O. Pollak. On the addressing problem for loop switching. *The Bell systems Technical Journal*, 50(8):2495–2519, 1971.

[55] Patrick J. F. Groenen, Rudolf Mathar, and Willem J. Heiser. The majorization approach to multidimensional scaling for minkowski distances. *Journal of Classification*, 12:3–19, 1995. 10.1007/BF01202265.

[56] A. Guttman. *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[57] C.E. Helm. A multidimensional ratio scalling analysis of color relations. Technical report, DTIC Document, 1959.

[58] G.R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(5):530 – 549, May 2003.

[59] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 1–12, New York, NY, USA, 2009. ACM.

[60] Newso James and Dawn Song. Gem: Graph embedding for routing and data-centric storage in sensor networks without geographic information. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pages 76–88, New York, NY, USA, 2003. ACM.

[61] William Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*, volume 26 of *Contemporary Mathematics*, pages 189–206. American Mathematical Society, 1984.

[62] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31(1):7–15, April 1989.

[63] Brad Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, Boston, 2000.

[64] Robert Kleinberg. Geographic routing using hyperbolic space. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1902 –1909. IEEE, 6-12 2007.

[65] Leonard Kleinrock and Farouk Kamoun. Hierarchical routing for large networks: Performance evaluation and optimization. *Computer Networks*, 1(3):155–174, 1977.

[66] Y.-B. Ko and N.H. Vaidya. Geocasting in mobile ad hoc networks: location-based multicast algorithms. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pages 101 –110, feb 1999.

[67] Tamara G. Kolda and Brett W. Bader. Tensor decpompositions and applications. *SIAM review*, June 2008.

[68] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 181–192, New York, NY, USA, 2007. ACM.

[69] D. Krackhardt. Cognitive social structures. *Social Networks*, 9(2):109–134, 1987.

[70] D. Krioukov, K. Fall, and X. Yang. Compact routing on internet-like graphs. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 1, page 219, 7-11 2004.

[71] Dmitri Krioukov, Fragkiskos Papadopoulos, Marián Bogu ná, and Amin Vahdat. Greedy forwarding in scale-free networks embedded in hyperbolic metric spaces, 2009.

[72] J.B. Kruskal. Nonmetric multidimensional scaling: A numerical method. *Psychometrika*, 29(2):115–129, 1964.

[73] Fabian Kuhn, Roger Wattenhofer, Yan Zhang, and Aaron Zollinger. Geometric ad-hoc routing: of theory and practice. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, PODC '03, pages 63–72, New York, NY, USA, 2003. ACM.

[74] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Asymptotically optimal geometric mobile ad-hoc routing. In *Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, DIALM '02, pages 24–33, New York, NY, USA, 2002. ACM.

[75] Ben Leong, B. Liskov, and R. Morris. Greedy virtual coordinates for geographic routing. In *Network Protocols, 2007. ICNP 2007. IEEE International Conference on*, pages 71 –80, 16-19 2007.

[76] C.R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *Selected Areas in Communications, IEEE Journal on*, 15(7):1265 –1275, sep 1997.

[77] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 577 –591, nov 1994.

[78] Yun Mao, Feng Wang, Lili Qiu, Simon S. Lam, and Jonathan M. Smith. S4: Small state and small stretch routing protocol for large wireless sensor networks. In *In Proc. of the USENIX NSDI Conf*, 2007.

[79] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[80] Radhika Nagpal, Howard Shrobe, and Jonathan Bachrach. Organizing a global coordinate system from local information on an ad hoc sensor network. In *Proceedings of the 2nd international conference on Information processing in sensor networks*, IPSN'03, pages 333–348, Berlin, Heidelberg, 2003. Springer-Verlag.

[81] Badri Nath and Dragoş Niculescu. Routing on a curve. *SIGCOMM Comput. Commun. Rev.*, 33(1):155–160, January 2003.

[82] Dragos Niculescu and Badri Nath. Trajectory based forwarding and its applications. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, MobiCom '03, pages 260–272, New York, NY, USA, 2003. ACM.

[83] Christos H. Papadimitriou and David Ratajczak. On a conjecture related to geometric routing, 2005.

[84] F. Papadopoulos, D. Krioukov, M. Bogua, and A. Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *INFOCOM, 2010 Proceedings IEEE*, pages 1 –9, 2010.

[85] William A. Pearlman. *Digital Signal Compression: Principles and Practice*. Cambridge Univ Pr, Oct 2011.

[86] David Peleg. Proximity-preserving labeling schemes. *Journal of Graph Theory*, 33(3):167–176, 2000.

[87] John C. Platt. Fastmap, metricmap, and landmark mds are all nystrom algorithms. In *Microsoft Research*, 2005.

[88] The NDN project team. Named data networking (ndn) project. Technical report, PARC, October 2010.

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

[90] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 161–172, New York, NY, USA, 2001. ACM.

[91] Sylvia Ratnasamy, Brad Karp, Scott Shenker, Deborah Estrin, Ramesh Govindan, Li Yin, and Fang Yu. Data-centric storage in sensornets with ght, a geographic hash table. *Mob. Netw. Appl.*, 8(4):427–442, August 2003.

[92] Y. Rekhter and T. Li. An architecture for ip address allocation with cidr. RFC 1518 (Historic), September 1993.

[93] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[94] V. Roth, J. Laub, J.M. Buhmann, and K.R. Müller. Going metric: Denoising pairwise data. *Advances in Neural Information Processing Systems*, 15:817–824, 2002.

[95] Volker Roth, Julian Laub, Motoaki Kawanabe, and Joachim M. Buhmann. Optimal cluster preserving embedding of nonmetric proximity data. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 25:2003, 2003.

[96] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[97] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

[98] Satu Elisa Schaeffe. Graph clustering. *Computer Science Review*, 1:27 – 64, 2007.

[99] B.S.W. Schröder. *Ordered sets: an introduction*. Birkhauser, 2003.

[100] Yi Shang, Wheeler Ruml, Ying Zhang, and Markus P. J. Fromherz. Localization from mere connectivity. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '03, pages 201–212, New York, NY, USA, 2003. ACM.

[101] Y. Shavitt and T. Tankel. Big-bang simulation for embedding network distances in euclidean space. *Networking, IEEE/ACM Transactions on*, 12(6):993 – 1006, dec. 2004.

[102] Blake Shaw and Tony Jebara. Structure preserving embedding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 937–944, New York, NY, USA, 2009. ACM.

[103] Aleksandrs Slivkins. Distance estimation and object location via rings of neighbors. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, PODC '05, pages 41–50, New York, NY, USA, 2005. ACM.

[104] Guang Tan, Marin Bertier, and Anne-Marie Kermarrec. Convex Partition of Sensor Networks and Its Use in Virtual Coordinate Geographic Routing. In *INFOCOM 2009*, Rio de Janeiro Brésil, 04 2009.

[105] Liying Tang and Mark Crovella. Virtual landmarks for the internet. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, IMC '03, pages 143–152, New York, NY, USA, 2003. ACM.

[106] Mingdong Tang, Hongyang Chen, Guoqing Zhang, and Jing Yang. Tree cover based geographic routing with guaranteed delivery. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1 –5, May 2010.

[107] Robert Tarjan. Depth-first search and linear graph algorithms. In *Switching and Automata Theory, 1971., 12th Annual Symposium on*, pages 114 –121, oct. 1971.

[108] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

[109] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, SPAA '01, pages 1–10, New York, NY, USA, 2001. ACM.

[110] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, January 2005.

[111] P. F. Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. *SIGCOMM Comput. Commun. Rev.*, 18(4):35–42, August 1988.

[112] W. T. Tutte. Convex representations of graphs. *Proceedings of the London Mathematical Society*, s3-10(1):304–320, 1960.

[113] George Varghese. *Network Algorithmics,: An Interdisciplinary Approach to Designing Fast Networked Devices (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann, December 2004.

[114] J.T.L. Wang, Xiong Wang, D. Shasha, and Kaizhong Zhang. Metricmap: an embedding technique for processing distance-based queries in metric spaces. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 35(5):973 –987, oct. 2005.

[115] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[116] K.Q. Weinberger and L.K. Saul. Unsupervised learning of image manifolds by semidefinite programming. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–988 – II–995 Vol.2, june-2 july 2004.

[117] Bernard Wong, Aleksandrs Slivkins, and Emin Guen Sirer. Meridian: A lightweight network location service without virtual coordinates. In *In SIGCOMM*, pages 85–96, 2005.

[118] E.P. Xing, A.Y. Ng, M.I. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. *Advances in neural information processing systems*, 15:505–512, 2002.

[119] L. Yang and R. Jin. Distance metric learning: A comprehensive survey. *Michigan State University*, pages 1–51, 2006.

[120] X.. Yang, D.. Clark, and A.W. Berger. Nira: A new inter-domain routing architecture. *Networking, IEEE/ACM Transactions on*, 15(4):775 –788, aug. 2007.

[121] Huaming Zhang and Swetha Govindaiah. Greedy routing via embedding graphs onto semi-metric spaces. In Mikhail Atallah, Xiang-Yang Li, and Binhai Zhu, editors, *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, volume 6681 of *Lecture Notes in Computer Science*, pages 58–69. Springer Berlin / Heidelberg, 2011.