Link Loss Tomography and Topology Synthesis

Rhys Bowden

Thesis submitted for the degree of

Doctor of Philosophy

in

Applied Mathematics

at

The University of Adelaide

Discipline of Applied Mathematics

School of Mathematical Sciences

THE UNIVERSITY
of ADELAIDE

January 22, 2015

Contents

Abstract

Accurate and timely performance data are of vital importance for network administration. However, modern networks are so large and transmit such enormous quantities of data that collecting the complete measurements can be wildly impractical. *Network tomography* uses the measurements that *are* available to infer underlying performance statistics. In this thesis we consider estimating average packet loss rates on links from more easily collected path measurements. Most such work has concentrated on tree-networks, but here we consider the problem on a general network where the problem is typically underconstrained. In that context we need some criteria to select a solution from the infinite set of possibilities. Here we exploit the compressive sensing assumption of sparsity. However, although the assumption of sparsity makes a great deal of sense in this context, the standard conditions required for results in compressive sensing theorems do not hold for realistic routing matrices. We show that despite this, the underlying techniques can still provide useful answers. What's more, we show that the apparently inconvenient structure of routing matrices can actually help in solving the problem efficiently. We provide CTD, a new algorithm for finding sparsest solutions that is orders of magnitude faster than one of the standard compressive sensing algorithms, and which provides more certainty. We also provide a version of CTD for working with a limited number of measurements, CTDn.

The success of a tomography algorithm often depends upon the underlying network topology. To test CTD we use two sources of topologies: the Internet Topology Zoo and synthetically generated topologies. We propose a new method for topology synthesis, Combined Optimisation and Layered Design (COLD), that mirrors the real-life design process of a data network. Since real data networks are designed, they are in some sense optimized to fulfil a function. However, strict mathematical optimisation is rarely performed at the router lever; rather the PoP-level is typically the one being optimized, because it is both more stable and less intricate. Once the PoP-level network is determined, the router-level network can be created using a templated design process, increasing regularity and aiding management and debugging. COLD mirrors this process by having two layers: PoP-level optimisation of realistic economic constraints subject to demand; and then router level synthesis with a templated design process

using graph products. We show COLD produces sensible, varied yet controllable output with a clear relationship to the relatively few input parameters.

We test CTD and CTDn with both Topology Zoo topologies and COLD-generated topologies. Because the COLD process can be controlled to generate a wide variety of topologies this allowed us to test the sensitivity of CTD and CTDn to the form of the underlying topology. We show that CTD and CTDn perform well across a wide variety of topological structures and forms of measurement, while relying on less detailed information than previous approaches.

Declaration

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, and partner institution responsible for the joint award of this degree.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Signature..... *Date*.....

Acknowledgments

Many thanks to my supervisors, Professor Matthew Roughan and Professor Nigel Bean, for their time, for their excellent advice and expertise, for their patience and forbearance, and for being examples of the kind of academic I can only hope to be one day.

Thanks to my colleagues and collaborators. I've enjoyed working with all of you, and I hope we work together again in future.

Thanks to Mum and Dad for their support and love throughout my whole life, not just during my Ph.D. candidature.

Thanks to all my family and friends for making life so enjoyable.

And finally thanks to Jess for everything. Including being Jess.

Chapter 1

Introduction

The Internet is a gigantic global network of networks and computers that grows bigger and more complex each year. It is used for a wide variety of applications – the world wide web, email, voice over IP phone calls, teleconferencing, shopping, other monetary transactions, streaming video and sending huge data files, amongst others. Every different application you use, and every different website you visit requires your computer to communicate with a different computer, sometimes on the other side of the world. The information being communicated does not travel directly to the other computer on a wire, instead it follows a route through a series of intermediate computers in a variety of different places owned by a variety of different organisations. These intermediate computers are called routers. When something goes wrong in sending this information, where do we look for answers? There could be a fault in the local router, the copper wire between our wall socket and the local telephone exchange, or misconfiguration of computers belonging to our Internet Service Provider (ISP). There could be a fault or error with any of the transmission lines or routers that form the route from our computer to the other computer, potentially on the other side of the world. With so many places to check, how do we find where the fault lies?

These problems are mirrored for the owners and operators of networks that constitute the Internet. There is too much data travelling through even a moderately sized Internet Service Provider to keep track of it all. Efficient measurement, maintenance and debugging are greatly complicated by the huge amount of data flowing through each network. Furthermore, too many measurements can congest the system and cause performance degradation themselves. Problems can occur at any one of several different layers/protocols, as well as at many different locations.

As a network owner/operator problems can occur on your network, an adjacent network, or a network further away. The complication of inter-organisation cooperation often means that there is limited information with which to find faults or poor performance.

Pinpointing the physical and conceptual locations of the errors can be very difficult.

Network tomography is a field that deals with making the most of the limited measurements we *can* make. *Network link tomography* uses performance properties of paths through a network to infer the performance properties of individual links or routers in that network. "Performance properties" refers to statistics like the time taken for a message to traverse that link (delay) or the probability that a message fragment will be lost or discarded at that link (loss).

Different paths overlap in different places in the network. Network link tomography methods can use this to determine what is going on in the network. For example, if a group of paths were all having similar problems, and they had exactly one link in common, then we would not be surprised if that link was the cause of their problems. Mathematically speaking this can be viewed as an inverse problem. We have a model for the way link properties affect the path measurements, that is, we can calculate what path-based measurements we would make for any given set of link properties. However, we wish to work backwards and determine the link properties from the path-based measurements made.

Often in network link tomography there is not enough information to solve for all the link properties uniquely. That is, there are many different choices for link properties that would explain the measurements equally well. We need a means to differentiate between these choices of link properties. One popular method of distinguishing possible solutions is determining how likely each solution is, based on a combination of prior knowledge and the knowledge we gain from taking measurements. Unfortunately, sometimes when following this approach it is difficult to compute the solutions, and at other times the solutions are sensitive to the way we quantify the prior knowledge.

To aid in solving link tomography problems we enlist the help of a relatively new field of mathematics: *compressive sensing*. Compressive sensing deals with data that has been altered before being measured, so that only limited *views* or summary statistics of the data are available. There are many reasons why real-world information might be compressed in the process of measuring it. Sometimes the measurement devices only have limited access to the real-world structures, such as MRI machines scanning people, or astronomical measurements being blocked by clouds. Other times, technical constraints like power constraints for autonomous sensor arrays or storage constraints when sensing large real-time data streams prevent gaining access to all the desired information.

Compressive sensing faces a similar inverse problem to that faced in network tomography: how to recover the underlying data of which the measurements provide a limited view. Just as in network tomography, there are many possible data vectors consistent with the measurements. In compressive sensing, the data being sensed is *simple* in some

way – and this gives a way to find the data vector amongst all the other vectors that fit the measurements. A data vector is considered simple if it is *sparse* in some basis; that is, if it can be expressed as a (linear) combination of relatively few basis elements. The desired data vector is the *sparsest* data vector consistent with the measurements.

In a well-functioning computer network, links with high delay or loss are relatively rare. This means the vector of link performance properties is sparse. Also, the path performance measurements are related linearly to the link performance properties (a requirement for compressive sensing). Consequently, compressive sensing provides a natural fit for network tomography. The only caveat is that the way in which path measurements are taken typically provides an inferior view of the link properties than is usual in the compressive sensing literature. This thesis presents a method, *Coherent Tomographic Deduction*, for mitigating the poor quality of these measurements and solving the link tomography problem. CTD takes advantage of the unique structure of the measurements to dramatically increase the efficiency of the algorithm.

The interaction between path performance measurements and link properties is highly dependent on the topology of the underlying network. Consequently, the success or failure of CTD can strongly depend on the underlying network topology. Thus, to test CTD thoroughly we need a set of network topologies on which to test it. While we have access to some real-life topologies transcribed from network-operator supplied maps, this is not enough. The only way to acquire enough network topologies to make accurate statistical inferences is to synthesize them.

Previous methods for topology synthesis are lacking for varied reasons. Some methods are concerned with matching one or two statistics and produce networks that don't even satisfy basic technical constraints (like router-throughput constraints, or being connected). Other methods fit a series of statistics very well, but are difficult to compute and risk producing almost no variety of output networks. Furthermore, they can produce a topology that is based purely on matching these graph-theoretic measures without a consideration for the underlying design process.

This thesis presents our own method for network topology synthesis, Combined Optimized Layered Design (COLD), which synthesizes a network based on the process used when designing a real network. It follows three steps: determination of a geographical and economic context, design of an optimized Point-of-Presence (PoP) level network in that context, and templated design to synthesize a router-level network from the PoP-level network. These produce a synthesized network rich in additional information, like link capacities and how many clients are serviced by each PoP. Further, the geographic and economic context can provide a rich variety of network topologies from the same design process. Tuning a small number of economic cost parameters can provide a wide variety of different types of network topology: topologies with a relatively large number of links, topologies with a relatively small number of links, or even topologies with a

few central high-traffic PoPs and many low-traffic PoPs nearer the edge of the network.

1.1 Chapter Summary

Chapter 2 is the Background. It explains the problem examined in this thesis in more detail; defines our notation; and summarizes the relevant literature. Chapter 2 is broken into three parts: a section focusing on network tomography, a section focusing on compressive sensing, and a section focusing on synthesizing topologies (in that order).

Chapter 3 presents a new topology synthesis algorithm, Combined Optimized Layered Design (COLD). COLD [1] generates a geographic context randomly, and takes economic constraints as the input parameters. It then uses a Genetic Algorithm to heuristically optimize the placement of links between PoPs to create a PoP-level network. The final stage of COLD is to produce a router-level network from the PoP-level network using Graph Products [2]. This process provides synthesized network topologies optimized from real-life constraints; but with a patterned structure that is common in real-world networks to allow efficient design, debugging and maintenance.

The latter part of Chapter 3 examines the relationship between the input parameters to COLD and the output topologies. It also compares COLD-generated topologies to real-life topologies from the Topology Zoo [3, 4].

Chapter 4 presents CTD, a new link tomography method incorporating compressive sensing. It utilizes aspects of the structure of routing matrices to perform fast size reductions of the tomography problem, prior to using compressive sensing methods. The reductions dramatically increase the speed of the overall algorithm, and provide information about which components of the solution vector are know to be accurate, and which are best-effort estimates.

Chapter 5 shows the results of extensive testing of CTD on topologies synthesized with COLD. We see that CTD solves the link tomography problem very successfully, and is limited more by fundamental limitations in the measurements than by its own shortcomings.

Chapter 6 shows a modification of CTD to deal with inexact (noisy) measurements, CTDn. This modification is then tested with both real-life topologies and synthetic topologies. CTDn has a low proportion of errors over wide variety of scenarios, especially considering the number of sources of substantial error.

Chapter 7 is the Conclusion.

1.2 Publication List

Following is a list of those publications completed during my candidature that are relevant to the content of this thesis.

[1] R. Bowden, M. Roughan and N. Bean, "COLD: PoP-level Topology Synthesis", to appear in *International Conference on emerging Network EXperiments and Technologies (CoNEXT)*, 2014.

[2] E. Parsonage, H. Nguyen, R. Bowden, S. Knight, N. Falkner and M. Roughan, "Generalized graph products for network design and analysis" in *IEEE International Conference on Network Protocols (ICNP)*, 2011.

[3] S. Knight, H. Nguyen, N. Falkner, R. Bowden and M. Roughan, "The Internet Topology Zoo" in *IEEE Journal on Selected Areas in Communications*, 2011.

The following publications were completed over the course of my candidature, but have little overlap in content with the thesis itself.

[4] R. Bowden, H. Nguyen, N. Falkner, S. Knight and M. Roughan, "Planarity of data networks", in *International Teletraffic Congress (ITC)* 2011.

[5] J. Sommers, R. Bowden, B. Eriksson, P. Barford, M. Roughan and N. Duffield, "Efficient network-wide flow record generation" in *IEEE Conference on Computer Communications (INFOCOM)* 2011.

[6] B. Eriksson, P. Barford, R. Bowden, N. Duffield, J. Sommers, and M. Roughan, "Basisdetect: A model-based network event detection framework" in *ACM SIGCOMM Conference on Internet measurement (IMC)*, 2010.

Chapter 2

Background

In this chapter, we explain the problem examined in this thesis in more detail; define our notation; and summarize the relevant literature. The chapter is broken into 3 major sections:

1. Network tomography – The central problem this thesis seeks to address is network link tomography. We summarize the relevant work from the field of network tomography, define the notation we will use and the problem we will address.

2. Compressive sensing – a field of study that concerns recovering signals that have been compressed before measuring apparatus has even been used to sense them. We apply methods from this area to the network tomography problem.

3. Network topology synthesis – Solutions to a network tomography problem can be sensitive to the underlying network topology. In order to test our method thoroughly we need a set of topologies on which to test it. While we have access to real-life topologies transcribed from network-operator supplied maps [3], this is not enough. The only way to acquire enough network topologies to make accurate statistical inferences is to synthesize them. In this section we explain why previous methods of network synthesis are unsatisfactory, and outline our goals in creating a new method for synthesizing realistic topologies that mirrors real-life design processes.

We start with the section on network tomography.

2.1 Network tomography

Network tomography concerns inference of network properties from indirect measurements. Often, such measurements are the only ones available due to expense, data volume, or impracticality of directly measuring the features of interest.

Network tomography covers a wide range of forms of inference: from inferring source-destination traffic (traffic matrices) from the traffic on individual links [7], to estimating router-level topologies of the Internet [8]. Here we are concerned with performance measurements, where a network operator would like to know performance statistics for each link in a network. However, there can be many thousands of links in a large network, so direct measurements are often impractical. Instead, one makes do with end-to-end measurements of performance along a set of paths through the network, and from these infers the performance on the links.

Network tomography takes its name from medical tomography, as the process is fundamentally very similar. In *X-ray computed tomography* (CT scans) X-rays are sent through the body, and absorbed to differing degrees by different bodily structures. Those that aren't absorbed are measured on the other side. This gives a flattened, 2 dimensional projection of internal bodily structures from the X-ray emitter on to the X-ray sensing apparatus. This process is repeated with the emitter and sensor rotated to a series of positions around the body. The combination of these many different 2-dimensional projections and some sophisticated signal processing techniques is used to build up a 3-dimensional picture of the inside of the body part being scanned.

Just like medical tomography, *network link tomography* uses a series of measurements sent through the network to build up a detailed picture of the internal status of the network. Each measurement "absorbs" loss or delay from multiple links, but the combination of the measurements can be used to infer the loss or delay of individual links.

Network tomography problems are often hard because they are underconstrained [9]. We usually don't have enough information to unambiguously determine the link performance statistics from the path measurements. When there are many possible states of the network that could produce a given set of measurements, solutions to link tomography problems require a way of distinguishing between all these feasible states; typically using side-information or regularization of some form. In our case we exploit *sparsity*. A *sparse* vector or matrix is one with very few non-zero values. In this case, sparsity means the vector of link statistics that we are trying to infer has very few non-negligible values, *i.e.,* very few links with substantial loss or delay.

Network tomography is a well developed field at this point of time (see for example [10–12]). However, the vast majority of performance tomography has concentrated on trees. Trees are a natural topology to be investigating, since the set of measurements

from one source will induce a tree topology[1]. In that setting, it is possible to develop fast, recursive algorithms [11, 13], and to employ regularization such as sparsity relatively easily. For instance, Duffield [14] uses sparsity to address a problem he calls *binary performance tomography*. In binary tomography, links are classified as either good or bad, usually based on whether they exceed certain loss or delay thresholds. While this is a dramatic simplification, such simplifications are frequently necessary in network tomography. Approaches with complex statistical models can rapidly become computationally infeasible when used on general networks with tens or hundreds of nodes. The defining property Duffield uses is that bad paths must have at least one bad link on them. In the strongly separable case, there are never bad links on good paths (that is, bad links always make the path bad). He then uses the crucial assumption that bad links are relatively rare, and so the most likely explanation for a set of measurements will have as few bad links as possible. In a tree, he shows that the sparsest solution will be the one with "bad" links as high as possible in the tree. Arya and Veitch [15] also consider this link tomography problem on (unicast) trees but utilize the numerical levels of loss along paths, rather than simply classifying a path as "good" or "bad". They use ℓ_1-minimization to provide a solution both when exact loss rates are known, and also when measured levels of path loss are only approximate.

However, many networks are not trees. Some work has looked at combining measurements from multiple tree-like views of the network [16], but the approach meets immediate difficulties. At an intuitive level we can see that it would be hard to use sparsity in the same way because there is no longer a "top" of the tree towards which we can push loss or "bad" links.

There are other ways to prevent the network tomography problem remaining undetermined. One approach is to use multicast probes [17, 18]. In environments without multicast enabled, this approach can be emulated using a string of unicast probes sent back-to-back – exploiting the strong temporal correlation of loss events [11, 19, 20]. Another approach is to use several periods of probing to determine the covariances of the path loss rates, then use this to determine the variances of the link loss rates. Links with very low variance in loss rate are then assumed to have negligible loss rates, and this can greatly reduce the number of unknown links, making the tomography equations determined [9, 21]. In this work, we will avoid relying on multicast or upon accurate estimations of covariance statistics, removing the need to have several different probing rounds. Using sparsity for regularisation requires only an estimate of the average loss rates.

[1] under common assumptions about single-path, hierarchical routing.

2.1.1 Tomography notation

We consider a network described by the graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of (undirected) edges. The number of links is $|\mathcal{E}| = n$, and we have m measurements, each across a path q_j through the graph. A *path* q from node a to node b is a sequence of consecutive edges $e_1, \ldots, e_k \in \mathcal{E}$ starting at a and ending at b.

We consider here the problem of inferring the loss probabilities on links across a network from path measurements, but it is more convenient mathematically to consider transmission probabilities. We note that transmission probabilities are the complement of the loss probabilities, and define

$$\begin{aligned} t_i &= \text{link transmission probability for edge } i = 1, \ldots, n, \\ p_j &= \text{path transmission probability for path } j = 1, \ldots, m. \end{aligned}$$

When losses on different links are independent the two are simply related by

$$p_j = \prod_{i \in q_j} t_i, \quad \text{for all } j = 1, \ldots, m.$$

Network tomography is easiest when we have a linear relationship between variates, and so we take the log of the above equations, and write them in matrix form as

$$\boldsymbol{\rho} = A\boldsymbol{\tau}. \tag{2.1.1}$$

where

$$\begin{aligned} \tau_i &= -\log t_i, \quad i = 1, \ldots, n, \\ \rho_j &= -\log p_j, \quad j = 1, \ldots, m, \end{aligned}$$

and A is the $m \times n$ routing matrix defined by

$$A_{j,i} = \begin{cases} 1 & \text{if link } i \text{ is on path } j; \\ 0 & \text{otherwise.} \end{cases}$$

Note that we work with the negative of the log transmission probabilities for convenience, as this way $\tau_i, \rho_j \in [0, \infty]$. Zero loss probability on link i corresponds to $\tau_i = 0$, while $\tau_i = \infty$ corresponds to 100% loss probability, and similarly for ρ_j with respect to path performance. Since τ_i and ρ_i are increasing with increasing loss, and they are zero when the loss is zero, for convenience we will hereby refer to them as *loss coefficients*.

In addition to the independence between links already mentioned, we assume that the routing matrix A is known and static over the measurement period. Our approach is agnostic as to whether multicast or unicast measurements are used. We do not exploit

correlations in loss in specific probes, but rather consider the statistical averages over a set of measurements, and so avoid the need to have probes that are strongly correlated in time. We also avoid the need to have multiple measurement periods, to correlate losses on paths, as in [22]. We assume that Equation 2.1.1 is consistent, that is that there exists a real solution to find. This corresponds to assuming there is no noise in the measurement system. This is an assumption we will weaken in Chapter 5.

The goal here is to find the $n \times 1$ vector $\boldsymbol{\tau}$ from the m path measurements $\boldsymbol{\rho}$. The problem would be simple if A were full-rank, as the problem is then just one of solving linear equations. However, A is rarely full-rank. Even when $m > n$, it is common that redundancy in the measurements leads to an underconstrained set of equations with infinitely many solutions. To solve this we need some side-information, usually in the form of a model, or additional assumptions about the loss process. In this work we exploit sparsity, though not in the way it has been used in the past.

In this thesis we attack the problem of network-performance tomography on a general network. We also exploit sparsity, though we do it without reducing the problem to a binary (good/bad link) problem. Instead, we use techniques and results from the ideally suited field of *compressive sensing*. Compressive sensing is a new field used in areas where data is compressed before being measured – either because the data is too big to store or even sense, or because of the way the sensing apparatus operates. Compressive sensing can be applied to areas as varied as autonomous sensor arrays, single pixel cameras, infra-red cameras and CT scanners.

2.2 Compressive sensing

In compressive sensing, each measurement $y_i \in \mathbb{R}$ is a linear combination $\boldsymbol{\phi}_i \mathbf{x}^*$ of the desired data $\mathbf{x}^* \in \mathbb{R}^n$, $i = 1, ..., m$; with $m << n$. Now if we let $\Phi = [\boldsymbol{\phi}_i]_{i=1}^m$ be the m by n measurement matrix, and $\mathbf{y} = [y_i]_{i=1}^m$ be the m by 1 vector of measurements then $\mathbf{y} = \Phi \mathbf{x}^*$. Since $m << n$ there are infinitely many vectors $\mathbf{x}$ consistent with the measurements $\mathbf{y}$. Normally this would prevent recovery of the data, but when the data is simple (*sparse* or *compressible* under some linear transform), then under certain conditions the data can be recovered uniquely (or in the case of compressible data, approximated well).

Despite being a new field, compressive sensing has a deep body of literature. There are three main questions this literature seeks to address.

1. How can we find a good dictionary for a specific application? A good dictionary is a set of vectors such that every signal vector is a linear combination of relatively

few elements of the dictionary. Finding one can potentially be very difficult, although for this work the dictionary is trivial.

2. What measurement matrices can be used for compressive sensing? How efficient are different types of measurement matrices?

3. What techniques can be used to recover the data $\mathbf{x}$? How efficient are these methods? How well does each method recover the data?

Works in the area address one or more of these questions, and the answers to each question are frequently inter-dependent. In our case, the answer to the first question is simple: since a well-run network has relatively few poorly functioning links, the data vector we wish recover (the transmission probabilities) is already sparse. If we were to try to recover delay instead, we could assume that most intra-network links have very low delay relative to the poorly functioning links we are trying to find.

The answers to the second and third questions are not so simple, and require some background. We take a step back and start by considering what it means for a data vector to be sparse.

Definition 1. *We say that a vector* $\mathbf{x}$ *is* S-sparse *iff* $||\mathbf{x}||_0 \leq S$ *where* $||\cdot||_0$ *denotes the* ℓ^0 *norm, i.e.,* $||\mathbf{x}||_0$ *counts the number of non-zero elements of* $\mathbf{x}$.

The lower $||\mathbf{x}||_0$ is, the sparser (less dense) $\mathbf{x}$ is. Now that we have defined sparsity, we can provide the absolute minimum requirements for a measurement matrix. For successful recovery of $\mathbf{x}^*$ to be possible, it must be the unique S-sparse vector satisfying $\mathbf{y} = A\mathbf{x}$ for some S. The following is a well known result justifying the idea that introducing sparsity can induce a unique solution.

Theorem 1 (Uniqueness [23])**.** *Let* Φ *be a matrix with every set of* $2S$ *columns linearly independent. Then for any measurement vector* $\mathbf{y}$*, the system* $\mathbf{y} = \Phi\mathbf{x}$ *has* at most one S*-sparse solution for* $\mathbf{x}$.

That is to say, if we know that our routing matrix obeys the above property, and the solutions of interest are sufficiently sparse (S-sparse), then we know there will be a unique solution corresponding to our measurements. This doesn't give us any information about how to efficiently find $\mathbf{x}$, however.

We might try finding $\mathbf{x}$ by simply solving the constrained optimisation problem:

$$\min_x ||\mathbf{x}||_0 \text{ subject to } \mathbf{y} = \Phi\mathbf{x}. \tag{2.2.1}$$

In other words "find the sparsest solution that is consistent with the measurements taken". In general a direct solution to this problem is a combinatorial optimisation

problem, and the worst case run-time grows very quickly with respect to the sparsity of $\mathbf{x}$. The compressive sensing literature now has a wealth of algorithms for solving these problems efficiently [24–27].

Perhaps the most popular method has been ℓ_1-minimization. The ℓ_1-norm of $\mathbf{x}$ is defined as $||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i|$. Instead of finding the sparsest data vector, ℓ_1-minimization finds the data vector of lowest ℓ_1-norm that is consistent with the measurements. The optimization problem 2.2.1 is replaced with

$$\min_x ||\mathbf{x}||_1 \text{ subject to } \mathbf{y} = \Phi\mathbf{x}. \tag{2.2.2}$$

This works because, roughly speaking, the ℓ_1-norm is the best convex approximation to the ℓ^0-pseudonorm. Solving (2.2.2) is simply a matter of solving a linear program (with $2n$ variables), far easier to achieve in general than solving (2.2.1).

An early result concerning this method was the following [23]: suppose that Φ is the concatenation of two orthonormal bases, so that $n = 2m$. Now suppose that the *coherence*, the maximal inner product between any pair of columns of Φ, is at most M. Suppose that $||\mathbf{x}^*||_0 < N$, and $\mathbf{y} = A\mathbf{x}^*$. If $N < M^{-1}$, $\mathbf{x}^*$ is the unique optimally sparse data vector satisfying $\mathbf{y} = A\mathbf{x}$. Let $\mathbf{x}_1$ be the solution to (2.2.2). If $N \leq (1 + M^{-1})/2$ then $\mathbf{x}^* = \mathbf{x}_1$. That is, we can find the sparsest data vector consistent with the measurements simply by solving a linear program.

The coherence is a crude measure of how well a matrix functions as a measurement matrix. With improving measures it was possible to greatly reduce the number of required measurements for signals of a given sparsity.

One such measure is RIP [28]. Roughly speaking, RIP measures how closely small subsets of columns of Φ act like orthonormal sets. Using RIP it is possible to place much stronger guarantees of performance on measurement matrices. Unfortunately, in network tomography we don't get to choose our measurement matrices, they are the routing matrices. Before we can apply compressive sensing results we have to test whether routing matrices make good measurement matrices. The first hurdle to overcome is that routing matrices are $0-1$-matrices, whereas most sensing matrices in the compressive sensing literature are real valued with a wide variety of entries between 0 and 1. Reducing the variety of entries reduces the efficiency of the measurement matrix, however Chandar [29] showed that 0-1-matrices can still function as measurement matrices. Unfortunately routing matrices don't even have columns with similar norm, nor do they satisfy RIP in general. They don't even have a low coherence between the columns. Despite this, we see in Chapter 4 that we can still use compressive sensing methods to recover the desired performance vectors, albeit with some modifications.

In Chapter 4 we present methods for performing network link tomography using compressive sensing. However, since the success of these methods can depend on the underlying network topology, we need topologies on which to test the methods we present. We use two sources for network topologies:

1. Router level topologies from the Network Topology Zoo [30], a collection of network topologies transcribed from maps provided by network operators and ISPs.
2. Synthetic network topologies synthesized with a router-level method detailed in Chapter 3.

Synthetic topologies generated using network topology models serve several purposes that the Topology Zoo topologies cannot. Synthetic topologies provide an arbitrarily large source of topologies allowing accurate estimation of the performance of tomography algorithms, and they allow control and tunability of the test topologies, enabling understanding of what features effect the success of the tomography algorithms. However, before we can synthesize topologies we need a *model* for data network topologies.

2.3 Network topology modelling

Network topology modelling is the study of the fundamental connectivity properties of computer networks. Typically networks and their connectivity structures are abstracted as mathematical graphs ("topologies") with nodes representing network elements and edges representing the links between them. Network topology research encompasses a diverse range of networks including social networks, networks of webpages and hyperlinks, biological networks, neural networks, transport networks, physical networks and virtual networks. This work addresses data network topologies, for networks in the Internet. Modelling and understanding data network topologies is vital since the ability for a data network to deliver data successfully or efficiently can depend on the topology of that network.

A key goal of network topology modelling is *topology synthesis*, creating artificial topologies with *similar properties* to real network topologies. The resultant topologies have most commonly been used in network simulation and emulation in order to test new networking algorithms and protocols whose properties and performance often depend on the structure of the underlying network. There are several good reasons for synthesizing network topologies rather than using real network topologies:

- Data on real network topologies can be hard to come by, due to commercial confidentiality or practical reasons.

- Often a very large number of network topologies are needed to get accurate statistical measures of effectiveness (like confidence intervals) when testing a new algorithm or protocol [5, 31, 32].
- Synthesized networks offer a greater degree of control, granting the ability to determine the crucial features of a network upon which an algorithm or protocol depends.
- Tools such as traceroute that are typically used to measure Internet topologies are not designed for this task, and can frequently produce inaccurate, misleading or incomplete data with strong measurement biases.
- Accurately producing synthetic topologies leads to greater understanding of the features important in the formation and evolution of networks.

The history of network topology research is littered with discarded topology models. One of the earliest models, used to represent both router-level networks and Autonomous System-level (AS-level) networks was the Erdös-Rényi graph [33]. Erdös-Rényi graphs are random graphs generated from two parameters: the number of nodes n, and another parameter proportional to the number of links.

Erdös-Rényi graphs come in two types. To construct an undirected Erdös-Rényi graph, $\mathrm{ER}_1(n, p)$, start with n nodes. Then, independently for each of the $\binom{n}{2}$ pairs of nodes, place a link there with probability p.[2] The defining feature of Erdös-Rényi graphs is that the existence of a link is independent of every other link, and that all potential links have the same probability of existing.

Erdös-Rényi graphs satisfied an early need for a stochastic method of generating topologies to use when testing new network protocols and algorithms, while allowing for a small degree of control over the output graphs (i.e. control over the number of nodes and links).

Waxman [34] followed on from Erdös-Rényi graphs by adding some spatial structure to this model – the intuition that longer links are more expensive and hence less likely to occur. Unlike Erdös-Rényi graphs, in a Waxman graph, each link has a different probability of occurring. The probability that there is a link between node u and node v is $P(u, v) = \beta e^{-d(u,v)/L\alpha}$, where $d(u, v)$ is the distance between nodes u and v, L is the maximum distance between two nodes, and α and β are parameters in $(0, 1]$. That paper [34] was an early attempt to add some realism to random graph models, but it

[2] These are Erdös-Rényi g̃raphs of type 1. To construct an undirected Erdös-Rényi graph of type 2, $\mathrm{ER}_2(n, m)$ start with n nodes again. Then randomly choose m unordered pairs of nodes and place a link between each pair. We will only consider Erdös-Rényi graphs of type 1, but Erdös-Rényi g̃raphs of type 2 have very similar properties; they are Erdös-Rényi graphs of type 1 conditioned on the number of links.

suffered from two major difficulties that are still important issues for topology synthesis today:

1. How to choose the parameters α and β and the placement of the nodes.
2. It could still produce networks that were totally unrealistic, *e.g.,* disconnected networks. In fact, some choices of parameters made this highly likely.

Erdös-Rényi and Waxman graphs are early examples of a style of topology synthesis that is still very prevalent today – so-called *random graph models*. Typically random graph model papers focus on an important measure (often one or more graph statistics). Authors then present an elegant and parsimonius algorithm for stochastically generating graphs that can controlled to match any input topology in this measure. There is an extensive literature on random graphs (*e.g.* [35–41]) – they are appealing because even though they are built from a multiplicity of simple interactions they display high-level properties *i.e. emergent* behavior. We wish to preserve their simplicity and avoid using a large number of parameters as this has many detrimental effects.

When real data-networks were observed in detail it was noted that they had structure that was not well represented in simple random graphs. The next generation of innovations produced *structural topology generators* [35], these are based on the idea that a topology generator should reflect the obvious hierarchical structures visible in real networks. They also guaranteed connectivity, which Waxman graphs did not. The Georgia Tech Internetwork Topology Models (GT-ITM) incorporated all these features.

Structural topology generators were widely used until they were unable to explain new large-scale measurements of the Internet [40, 42–44]. Measurements of node degree distribution suggested that this distribution was heavy-tailed, following a power-law distribution [44], irreconcilable with the structural models of the time.

Attention then focused on finding random graph models that could explain this distribution [36–41, 43, 45]. These approaches were appealing in the way their simple local rules led to global properties. Many of the models were intended to mimic the growth of a network. Sometimes it was even inferred that because a model generated synthetic networks with power-law degrees, then networks evolved according to the model.

Criticisms soon appeared, ranging from comments about the problems with the data and its interpretation [46], to the fact that there are many possible networks with a power-law distribution, and only some of these satisfy other basic technical requirements for networks (*e.g.* port/throughput constraints on routers) [47–50]. Li *et al.* [51] use the *entropy* of a graph to make it clear that earlier methods will almost never generate networks that can carry reasonable quantities of traffic. They point out that large networks designed to carry traffic efficiently will (at the router level) have low-degree,

high-bandwidth nodes at the core, and higher degree nodes at the edge to aggregate traffic. While [51] provides a clear heuristic structure that router-level networks must obey if they are to exist in the real world, it does not give an accompanying method for synthesizing large quantities of example topologies.

Degree-series-based methods tackle the problem of synthesis by providing a general and extremely powerful framework for characterizing graphs, dK-series. When defining a dK-series, each node of a connected graph G is labeled with its node degree. The dK-distribution of G is the number of occurrences of each possible labeled connected subgraph of G of size d, where subgraphs are considered isomorphic if their labels and edges match.

The 0K distribution for a graph is simply its average node degree, and the 1K distribution is the node-degree distribution, both commonly studied when synthesizing topologies. The 2K-distribution is used to replicate the commonly seen *assortativity* statistic, as well as the entropy statistic used in [51]. The 3K-distribution determines the *clustering* of a graph. A dK-series for a graph G with n nodes is this sequence of dK-distributions for $d = 0, ..., n$. As d increases, the dK-distributions contain progressively more information without losing any; the 1K-graphs for G are contained in the 0K-graphs for G, the set of 2K-graphs is contained in the set of 1K-graphs, *etc.*

The dK-series are an elegant generalisation of the concepts of degree distribution to allow for increased fidelity with respect to an observed graph. It becomes very difficult to synthesize random dK graphs for $d > 3$, but a much more serious problem is that the dK-series hides an incredibly detailed characterisation of a graph G. A dK distribution isn't just a single statistic, it is a huge list of the number of occurrences of each labeled subgraph. Thus, although conceptually simple, the dK-series is very far from simple in practical terms. Figure 2.1 illustrates the rapid growth in the number of parameters with both d and the number of nodes n. Note that even for $d = 3$, we quickly approach the point where there are more parameters than edges in the graph. That is, the dK sequence has more "information" than the graph it is derived from.

3K and 4K-series graphs successfully solve the problem: how can we generate random graphs that are extremely similar to a specific graph, for almost all commonly used statistics? Unfortunately, in doing so, they heavily constrain the possible output graphs, sometimes suppressing desirable variety. dK-series can so overconstrain the problem that there is only one possible graph that can be generated. This problem can be hidden by the graph isomorphism problem – it is difficult to determine if two graphs are isomorphic (*i.e.,* effectively identical).

The issue is illustrated in Figure 2.2, which shows a simple input graph, along with Erdös-Rényi with the same mean degree, and 3K-series graphs generated to match it. It should be obvious that the only possible 3K graph that can match the input is the

input itself, though this might not be so obvious if the outputs were not aligned as in the figure. Moreover, it is not just a question of the outputs all being isomorphic – even if this is not strictly true, it is possible to have outputs that have a large isomorphic subnetworks and hence only trivial differences. These are even harder to test for. Design-based approaches avoid this problem providing additional details for the networks (like link capacities, routing and node positions) as part of the synthesis process itself.

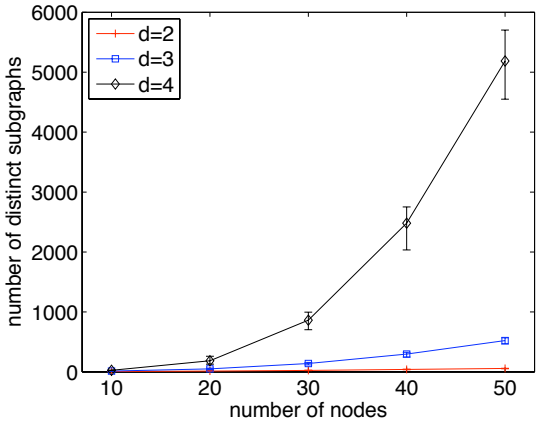


Figure 2.1: An example of how the number of parameters for dK-series grows rapidly both with the size of the graph and with d.

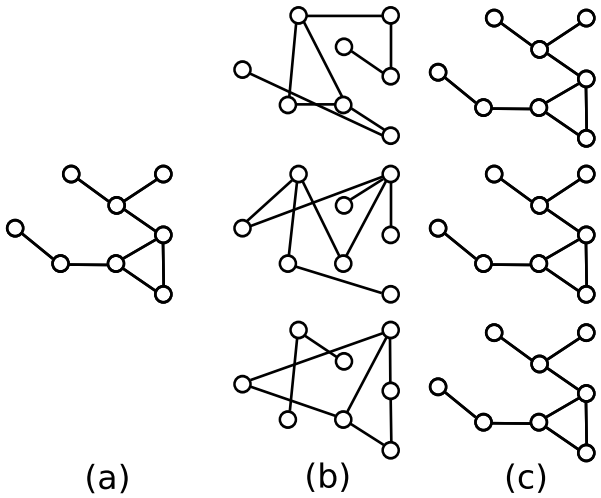


Figure 2.2: (a) A small example network. (b) Erdös-Rényi graphs based on that network - they all have the same number of links but in random places. One graph is not even connected and the others have very long shortest paths between some pairs of nodes. (c) graphs with the same 3K-distribution as in (a).

Graphs are high dimensional objects, and comparisons of "similarity" based on a limited set of abstract features can fail simple sanity tests [51]. For instance, Waxman graphs are not guaranteed to be connected. If enough statistics are included to ensure that

such tests are passed, we risk over-fitting the model, potentially resulting in a series of isomorphic graphs[3], of no statistical value in simulation.

While comparisons of statistics are inevitable, these are not the only criteria we need to consider. Designed networks have been shown to mimic observed statistics of real graphs, such as power-laws [51–53], but they use meaningful criteria in their design process, avoiding the production of patently unrealistic networks by generating network topologies that mirror real-life constraints and engineering goals.

Another underlying theme appears in the apparent conflict between random graphs and designed graphs. There is a clear contrast between the two competing approaches to modeling data networks. However, we do not need to choose: instead, we can steer a course between the two. Randomness is *needed*. We need to be able to generate a large number of statistically variable networks. However, we also need to incorporate structure and intelligent design.

Chapter 3 introduces an algorithm to incorporate randomness, structure and intelligent design: Combined Optimized Layered Design (COLD). COLD creates a random *context* for a network, then uses a (nearly) deterministic synthesis process that is motivated by the real goals, constraints and decisions that network engineers make in building a network.

Our approach is motivated by Li *et al.* [51] who note that the criteria for measuring similarity must originate with network operators. Their goals, constraints and decisions result in a network, and any method that ignores this in favor of overly simplified abstract synthesis risks generating unreasonable networks. We extend this idea from the HOT router-level synthesis described in [51] to multi-layered networks. Large data networks are not designed by a pure router-level optimization, they use hierarchy and templated design to simplify the design process [54–56]. The most common form of hierarchy is the division into PoPs (Points of Presence). Our approach, COLD (Combined Optimized Layered Design), is aimed at mirroring this design process.

In Chapter 3 we focus on the synthesis of PoP-level topologies, as the first step of a layered approach to generating more detailed router-level network topologies. We focus on the PoP-level because:

- this is the level at which many of the interesting problems lie (the generation of the router-level network from the PoP level can be easily accomplished using either existing probabilistic methods [35], or structural methods [57]);
- networks change less frequently at the PoP level than at the router level [58]; and

[3]There is no known polynomial time algorithm for testing if two graphs are isomorphic.

- the PoP level is the more interesting level for many activities [58], because it is less dependent on the details of protocol implementations, router vendor and model, and other technological details.

The last point is subtle but important. When using a network as part of a simulation, one would like to have a network that is *invariant* to the method being tested. If a network designer might change his/her network in response to a new protocol, say a routing or traffic engineering algorithm, then the test will be ambiguous if it uses existing networks as models. PoP-level networks are less sensitive to these details than router-level networks, because routers impose physical and technological constraints that are almost completely dependent on the details of the router vendor, model and even the version of software running on the router.

PoP-level optimization is substantially different to the router-level optimization considered in [51]. At the PoP level, we have different constraints – we are no longer limited by technological issues such as port numbers – and the optimization objectives are different, so a considerable part of Chapter 3 is concerned with framing, and then solving this problem (see Sections 3.3-3.7).

We go to considerable efforts to build a still simple but much more realistic design process. Our goal is to generate networks that closely resemble those running today and generate them in a *tunable* way able to replicate the wide variety of networks that are observed in the *wild* [3]. Although our philosophy is similar to [51], its implementation is very different: the optimization problem we solve is more complex, and we use techniques from graph theory to generate a multi-level hierarchy in a way that mimics the design patterns seen in real networks.

Chapter 3

Topology Synthesis

3.1 Introduction

The core goal of this thesis is to provide methods to address the network link tomography problem on general topologies. However, since the success or failure of these tomographic methods depends on the underlying network topology, we must provide a set of topologies on which to test our tomography methods.

There are several options here. We could use a set of 'real-life' topologies, either provided by operators or measured indirectly. Unfortunately, topologies determined with measurement tools are frequently inaccurate or incomplete, since the underlying protocols and tools used in measurement are rarely designed for topology discovery [46]. On the other hand, topologies provided by network operators have the advantage that they are likely to be accurate, but with the drawbacks that a limited number of operators are willing to provide this data, and that these topologies are not often in an easily usable form. The Internet Topology Zoo [3] solves the latter problem by processing publicly available images of network topologies; however we still have the former problem that there are relatively few topologies available.

To produce enough topologies to do a rigorous statistical testing of our tomography methods we need to turn to *topology synthesis*. The core goal of network synthesis is: *to take one or more network topologies, and produce a larger set of topologies that is similar in some fundamental way.* Historically a major difficulty has been that "similar" is very difficult to define. It may mean something completely different when considering virtual networks (*e.g.,* the Facebook Social graph) or physical networks (*e.g.,* data networks). For instance, a virtual network need not be connected, whereas a disconnected data network is broken. This thesis is concerned with data-communications networks.

Our approach is motivated by Li *et al.* [51] who note that the criteria for measuring similarity must originate with network operators. Their goals, constraints and decisions result in a network, and any method that ignores this in favor of overly simplified abstract synthesis risks generating unreasonable networks. We extend this idea from the HOT router-level synthesis described in [51] to multi-layered networks. Large data networks are not designed by a pure router-level optimization, they use hierarchy and templated design to simplify the design process [54–56]. The most common form of hierarchy being the division into PoPs (Points of Presence). Our approach is aimed at mirroring this design process.

When designing a topology synthesis algorithm there are several important challenges to be met:

1. Simulation requires us to generate a potentially large number of network topologies that are "similar", but varied enough to perform statistical analysis of results, *e.g.,* generating confidence intervals for performance estimates [31, 32].
2. A network's form is driven by real costs and technical constraints. For instance, it must be able to carry a given volume of traffic. If this is ignored, then the resulting network could be unreasonably expensive, or even impossible to construct [51].
3. Model parameters must be operationally meaningful. Parameters with only abstract meaning (*e.g.,* n^{th} degree distribution) are much harder to use in practice. For instance, it is often hard to scale abstract parameters correctly if one wishes to consider the effect of network growth. Parameters such as costs allow one to test network protocols given different tradeoff decisions by network engineers.
4. The approach should be *tunable*: it should be possible to control the output to see what effect the type of network has on a protocol or algorithm. For instance, one may wish to see the effect of a network becoming more highly connected or more clustered.
5. The model should generate a "network", not just a graph. Simulations often need details such as link capacity, distances, and routing. Ideally these should be generated as part of the model. If these details are generated after the topology synthesis then that additional process should be taken into account when considering the complexity of the method.
6. The model should be as simple as it can be – but not simpler. Simplicity has many virtues: it improves our intuitive understanding, reduces the complexity of parameter estimation, and prevents over fitting. There is a tension between "realism" and "simplicity" – determining the correct tradeoff between these is perhaps the most difficult of these challenges.

The algorithm we present here, *Combined Optimization and Layered Design* (COLD), satisfies these requirements. The key is choosing a synthesis process that parallels the

real network design used by many network engineers. Rather than designing the router-level network from scratch, we split the problem into layers. The first layer is designing the PoP-level network using heuristic optimization of economic objectives subject to physical and demographic constraints. The second layer is to use templated design to implement both physical router-level structures within PoPs (see [57] for example) and the connections between PoPs. This second step mimicks the highly structured and pattern-based methods recommended in basic texts on network management [54–56]. Likewise, simple heuristics can be used to create interdomain connectivity between networks if such is needed.

Our first step is to focus on the synthesis of the PoP-level. We start with PoP-level design because:

- This is the level at which many of the interesting problems lie (the generation of the router-level network from the PoP level can be easily accomplished using either existing probabilistic methods [35], or structural methods [57]).
- Networks change less frequently at the PoP level than at the router level [58].
- The PoP level is the more interesting level for many activities [58], because it is less dependent on the details of protocol implementations, router vendor and model, and other technological details.
- Optimizing at the router-level can quickly result in difficult or infeasible optimization problems, and even if successful can result in complex and difficult-to-manage networks.

The strength of COLD is that it incorporates common themes from network management, but is still as simple as possible while remaining flexible and tunable.

The generation process is deterministic. For any given context, the resulting network would be fixed. To generate the *stochastic variety* necessary for simulation, we randomize the *context* in which the network is generated, most notably the location of PoPs and the traffic matrix the network must carry. For any given real-world network the PoP locations are not random - they can be determined by a wide variety of factors including PoP and IXP locations for other networks, population density, and available building space. However, these factors themselves are different from place to place, and random to begin with. Since we are generating a body of realistic networks, rather than trying to duplicate any one network, starting with random locations for PoPs is ideal.

The result is an intuitive method for generating structured "designed" networks. The parameters are meaningful – they are costs, allowing them to be tuned to control the type of network generated: for instance, a newly formed network servicing a burgeoning market in a developing country may wish to provide connectivity as quickly and as cheaply as possible. As the market matures there is an incentive to increase the level of service by providing higher bandwidth, lower latency, or more reliability. Our process

can take these differing economic incentives or *planning variety* into account through *tuning* the input parameters.

We can generate a large number of different networks by randomizing the network context. The resulting networks come with all the details needed for simulation (*e.g.,* link capacities), and satisfy a range of simple standard network-engineering constraints. The model is easily extensible where needed.

3.2 Comparisons

In this section we evaluate and compare previous methods to the list of goals outlined in Section 3.1. Table 3.1 compares several different methods of network synthesis: Erdös-Rényi graphs, Waxman graphs, Power-law random graphs (PLRGs), Li *et al.*'s HOT graphs and Mahadevan *et al.*'s *d*K-series graphs against these criteria. While in many cases it is clear how these models perform against the criteria, some evaluations require further explanation.

Erdös-Rényi and Waxman graphs have a very simple model with few parameters. They are both simple and succeed in generating statistically varied graphs when given the same input. They also partially succeed in providing a mechanism to tune the output, though it is limited to controlling node degree for Erdös-Rényi graphs, and an additional notion of geographical distance dependence for Waxman graphs. Unfortunately, the parameters are of questionable physical meaning, and without modification they don't even meet simple technical constraints like connectivity. Further, these graphs do not generate any additional details such as link capacities.

Power-Law Random Graphs (PLRGs) [59] address the observed power-law node degree distribution of networks in measurement studies (at router- or AS-level). Their method is simple, involves few parameters, and the general class of such methods allows tuning of the average degree and the degree-distribution, but they have limited control of properties such as assortativity. Furthermore, these models have come under criticism [48, 51, 53] at the router level for violating technical constraints. At a deeper level, the model (preferential attachment) used to generate these networks has parameters that are, perhaps, meaningful for certain graph generation problems, but which certainly *aren't* meaningful for generating the types of networks considered here. PoPs do not "attach" to other PoPs according to a probability based on degree!

Li *et al.*[51] provide an outline of problems in previous topology synthesis techniques. They articulate engineering constraints that a real network must satisfy to function efficiently, and use these constraints to suggest a heuristic structure for real router-level networks. To aid their demonstration they introduce the entropy function for a graph

	ER	Waxman	PLRG	HOT	dK-series	COLD
1. statistical variation	✓	✓	✓	✓	×	✓
2. meets constraints	×	×	×	✓	P	✓
3. meaningful parameters	×	×	×	P	×	✓
4. tunable	P	P	P	P	×	✓
5. generates network	×	×	×	✓	×	✓
6. simple model	✓	✓	✓	✓	×	✓

Table 3.1: Table of comparisons between six synthesis methods against the criteria outlined in the introduction (in order). P refers to partially satisfying the requirement, for instance the dK-series can satisfy some constraints (*e.g.,* port numbers), but not others such as capacity constraints. Most of the models are only partially tunable because they can control one particular aspect of the network, *e.g.,* node degree, but not others.

(related to the assortativity) to clearly demonstrate the flaws of PLRG techniques. Their method has many appealing features, but the parameters and constraints of the model only represent a partial set of those that network engineers care about, and the design framework used does not mirror that used for the design of larger networks, though it may be reasonable for small networks.

Degree-series-based methods tackle the problem by attempting to extend the random graph methods to address the problems noted in places such as [51]. In the foremost examples of this approach, Mahadevan *et al.* [60, 61] provide a general and extremely powerful framework for characterizing graphs, dK-series. For more details on dK-series, see Section 2.3.

COLD satisfies the criteria listed above:

1. The networks are distinct by construction. We also present results showing the degree of variation through confidence intervals on observed statistics.
2. COLD uses techniques motivated directly by operator practice. One might argue that it is not exactly what any one operator does, but there is always a tradeoff between verisimilitude and simplicity.
3. The parameters are costs, which are intrinsically meaningful.
4. It is tunable, at least within the observed PoP-level networks of [3].
5. COLD generates more than just a series of connected nodes. It generates link capacities and distances along with routing.
6. The PoP-level model has only four parameters, and we show why at least this many are needed. In particular, the hub cost is needed to create networks that could match observed instances.

The last point raises one additional feature: extensibility. The optimisation algorithm facilitates extension because it is generally easy to add additional costs or constraints to the model. For example, COLD could naturally be extended to multiple ASes. Imagine the PoPs are in fact cities, in which different networks may have presence. PoP interconnects in same cities could then be assigned a cost, and we could run the optimization with respect to this additional cost.

Obviously, there are many other network synthesis approaches available, and we don't attempt to survey them all here. However, the examples above provide a sufficient cross-section to understand the limitations of the vast majority of synthesis techniques.

3.3 PoP-level Synthesis

We focus on synthesizing single data networks, such as a single network run by an Internet Service Provider. This is the level where design has the most influence, because the network is under the control of a small group of designers, all following a common process.

We use optimization to create our networks, but it is important to realize that few network designers are mathematicians or trained to use formal optimization tools. Moreover, the router-level design of a large network is very complex, involving many technical constraints (*e.g.,* port limits) that depend on varying details (*e.g.,* router models). Hence, networks are rarely designed from scratch – they evolve. Operators and managers try to optimize (by reducing costs, or improving performance) but usually do so heuristically.

More importantly, most of the optimization steps concern the PoP-level network. The internal design of PoPs (Points of Presence) is almost completely determined by simple templates [54–56], since the cost of internal links is much lower than inter-PoP links.

COLD mirrors this process by first generating an *optimized* PoP-level network, and then using templates to create a router-level map. In this section we explain the PoP-level optimization process, and defer the router-level network generation until §3.8. The guiding principles of the optimization are:

1. We must mirror the real-life process of designing a network.
2. The process needs to be *tunable*. Real networks come in a wide variety [3], determined by different underlying cost/benefit structures. We aim to be able to tune the input parameters of our process to replicate this wide range, either by choosing parameters to match a given set of networks, or by allowing these parameters to vary.

3. The optimization cost function and constraints must be as *simple* as possible; most notably they should have few parameters. The more detailed and complicated a cost function is, the less general and adaptable it is. If it becomes too complicated, it is hard to develop an understanding of the relationship between the input parameters and output networks, and hard to estimate parameters when needed.
4. The optimization cost function should be *meaningful*, and related to the criteria that are important to network engineers. Fabrikant *et al.* [52] show that it is possible to generate a wide variety of topologies by tuning an optimization process, but their cost function did not have a strong analogue to real-life costs. The meaning of the cost function also makes several tasks easier, such as extrapolating a network to examine what it might look like as it grows [32, 61] as it is difficult to know how purely abstract parameters should scale as a network grows.

There are several parts to an optimization scheme for synthesizing networks:

1. The *context* of the optimization problem, by which we mean the inputs to the problem: the PoP locations and the traffic matrix. The generation of these is described in §3.3.1.
2. The optimization problem itself. This includes the variables, the constraints and the optimization objective function. We minimize the cost of the network with the constraint that it can carry all of the expected traffic. We discuss this in detail in §3.3.2.
3. The algorithm for choosing an optimal network topology, which we describe in §3.3.3.

3.3.1 Context

It is not strictly correct to divide the area of network synthesis into *random graphs* and *designed* graphs as all the interesting models synthesize a random ensemble of graphs. The distinction lies in the way randomness is introduced, random graphs are typically constructed by repeated application of simple random rules, but designed approaches can introduce randomness through the inputs to the design process, *i.e.,* the context. In our problem the context consists of:

- the spatial locations of the nodes or PoPs; and
- the traffic matrix, giving traffic demands between each pair of PoPs.

We generate these randomly, so that each time we synthesize a new network we generate different positions and populations. Thus, even with fixed parameters we can generate an ensemble of networks guaranteeing that the generated networks are not the same.

We choose n PoP locations uniformly at random on the unit square. This is the simplest approach for generating their position. We use the physical distances between the PoPs to determine two components of the objective function used in the optimization.

We tested numerous alternatives including:

- Different region shapes - the shape of the underlying area makes little significant difference on the resulting networks, unless it is extremely long and skinny.
- Different distributions of PoP locations, primarily to make this distribution more bursty, but this also had only a small effect on the results (see §3.7 for details).

As these are *inputs* to the optimization step, there is no difficulty in using any model desired here and our tool provides the facility to allow this.

Our traffic matrix is created using a *gravity model*, proposed in multiple contexts [62–64], and tested [65] as a model for synthesizing traffic matrices. It suffers from identifiable flaws [64], but matches the distribution of real traffic matrices well [65] (our main requirement). Moreover, it is the maximum entropy model for traffic matrices under the circumstances described here [66]. The gravity model is populated by choosing a random population for each PoP. We tested two types of population model, the exponential model (populations were independent, identically distributed exponentials with mean 30), and the Pareto with shape parameters 10/9 and 1.5 (but the same mean), in order to test the impact of varying degrees of heavy tail on the results. Surprisingly, the effect on the inter-PoP topology of a heavy-tail in the traffic was small (see §3.7 for details) (although it might be larger in subsequent steps, e.g., mapping PoPs down to routers §3.8).

We prefer the simpler, exponential model in most of the subsequent work, though once again it is trivial to change this detail, and our tool provides this option.

3.3.2 Formulation of the optimization problem

Each candidate PoP-level topology is represented by an undirected graph $\mathcal{G}(N, E)$ with the same set of nodes N (the PoPs determined in the context step). However, each topology has a different set of edges, E (the links between PoPs). The variables in our optimization are thus the locations of the links in the PoP-level topology.

Each edge $i \in E$ is assigned a capacity w_i. The main constraint in the problem is that the capacities of the network are sufficient to carry the inter-PoP traffic (this implicitly requires the network to be connected). We do not include redundancy, port numbers or other complex constraints at this level. They are dealt with in the router-level construction §3.8.

Our cost function represents the cost of building the network and was chosen to be as simple as possible yet still able to approximate real-life objectives and produce a wide range of behavior. Its two components are link- and node-based costs, each described below.

Link cost

The cost for a link depends on many factors including the economic and geographical environment, existing infrastructure and networks, and other technical limitations. If these are modeled in too much detail we risk the model becoming less applicable to future networks and current networks in different environments. We keep the model as simple and general as possible, while allowing it to be tuned to produce a wide variety of networks.

To determine the cost for each link, given the set of links E, we start by determining the required capacity for each link. We perform shortest path routing using the physical distances between the nodes. We then set the required capacity for each link, w_i, by summing the total traffic demand for all the routes over that link, and multiplying by a constant factor to allow for variability in traffic[1].

The cost for link $i \in E$ is given by $C_i = k_0 + k_1\ell_i + k_2\ell_i w_i$, where ℓ_i is the length of link i and w_i is the bandwidth of link i and k_0, k_1 and k_2 are constants.

The link cost consists of three components:

0. k_0: The cost if the link exists.
1. $k_1\ell_i$: A cost for the physical length of a link; for instance, the cost of digging a trench for cabling or renting space in a conduit.
2. $k_2\ell_i w_i$: A bandwidth cost. This should approximately represent the cost of capacity.

Note that there is another way to interpret the total bandwidth cost, which is given by

$$\sum_{i \in E} k_2 \ell_i w_i = k_2 \sum_{r \in R} t_r L_r, \tag{3.3.1}$$

where R denotes the set of routes, L_r is the length of each route $r \in R$, and t_r is the traffic along route r. Thus this part of the cost is proportional to the traffic on each path (which is a fixed input to the problem) and the length of each path. This is the only part of the cost which is load sensitive, and hence which depends on the routing

[1] this factor is set to 3 by default, but can easily be changed. Any changes in this factor are equivalent to a corresponding scaling in the design constant k_2, defined below.

of traffic. The implication is that our cost function will be minimized by shortest-path routing for any given set of links E.

Node Cost

In our model, the number of nodes is fixed, so typically in optimization this would result in a constant node cost. However, we found that we needed a cost to differentiate types of PoPs.

Real networks show tremendous variability [3]. Some are meshy, and others more like a hub-and-spoke network. Optimization of link costs alone tends to produce meshy networks – for instance, when the k_2 cost is dominant, it results in cliques. We found experimentally that we could not get hub-and-spoke networks purely through optimizing against link costs.

Further examination of the networks in [3] also shows that they often have two classes of PoPs: *leaf* and *core*. Typically leaf PoPs all had only one link connecting them to the network (*i.e.,* they had node degree 1), and core PoPs had two or more links. Obviously, hub-and-spoke networks have more leaves than meshy networks.

The simplest and cleanest way of inducing leaf nodes in our optimization-based networks was to add a cost for *non-leaf* nodes. So each node j with degree$(j) > 1$ incurs a cost of k_3. This represents a *complexity cost*; a PoP with multiple connections to the outside world is more complicated to implement and maintain than one with only one connection. Complexity has a cost in real networks [67]. Managing a small PoP with only a single router, and/or single link is much simpler than a multi-router, multi-link PoP. We represent this in the simplest way possible, through a cost k_3 for each non-leaf PoP.

Optimization Problem

The optimization problem is therefore:

$$\min_{G(N,E)} \sum_{i \in E} (k_0 + k_1 \ell_i + k_2 \ell_i w_i) + \sum_{j \in N_C} k_3, \tag{3.3.2}$$

such that all traffic can be carried. Here G runs over all connected graphs on n nodes, and $N_C = \{j \in N \mid \deg(j) > 1\}$ is the set of core or hub nodes.

The costs k_0, k_1, k_2 and k_3 allow the process to be *tuned* to produce different types of topologies, by changing the relative importance of each part of the cost. To under-

stand how this trade-off works, we consider the impact of each component of the cost separately.

- k_0-*cost*: This cost depends on the number of links. Networks must be connected, so if this cost dominates, all the *spanning trees* are optimal solutions.
- k_1-*cost*: This is the cost for the total length of all links. If this cost dominates, then the optimum solution is a *minimum spanning tree*.
- k_2-*cost*: The k_2 cost can be interpreted as a cost for the length of the routes, see (3.3.1). Hence, when k_2 dominates the routes will be as short as possible, *i.e.,* the result will be a *clique* or *fully connected network*.
- k_3-*cost*: If this cost is dominant, the optimal network will have only one node with degree greater than one, *i.e.,* it will be a *hub-and-spoke* network.

Typically more than one cost will contribute, and so we will get a network that is a mixture of the above forms.

The components of the objective function are simple in themselves, and optimizing against any one is not difficult. However, the mixed optimization is not so simple. There are too many potential solutions for a complete enumeration for even moderate values of n. Moreover, the problem does not decompose into smaller problems, and the relaxation from an integer problem to the reals is not useful. Hence we solve it heuristically.

Guarantees that our solution is truly optimal are not necessary. This chapter is not about optimization, *per se*, but rather about an attempt to replicate the process of network engineering. Given the uncertainties in inputs such as the traffic matrix and cost model, network engineers are typically looking for a good solution, not the optimal, and they do so using their own heuristics.

3.3.3 Genetic Algorithm

Once we have formulated the optimization problem, we must decide on an algorithm for finding solutions. We use a heuristic search algorithm called a *Genetic Algorithm* (GA). It works by evaluating the objective function on an initial random population of candidate topologies. It then chooses the topologies with lower costs to be more likely to survive and pass on their "genes" to the next generation (either by crossover, mutation or cloning). The process is repeated for many generations. Since fitter topologies are more likely to survive, the overall population improves until all topologies are well-adapted to the environment, and the population reaches an almost-stable state.

While there are many candidate search algorithms, we choose to use a GA because it has the following properties:

1. *Flexibility*: GAs only require small adaptations to cope with changes to the objective function.
2. *Competiveness:* We do not need to find the true optimal solution, but we do need to find a good solution. One way to ensure this is to require that the GA's solution is at least as good as competitors. A key advantage of GAs is that we can include alternative solutions in the initial population, and thereby guarantee its solution is at least as good as these.
3. *Non-exclusivity:* For a given optimization problem, one run of a GA generates a population of solutions. The variation between these solutions can give a better idea of which characteristics are important in optimizing these topologies, and which are irrelevant. It also allows us to create multiple networks with the same context, potentially providing additional support for simulation where one wants a fixed context, but multiple topologies.

Of these, the first property has been most important here as it has allowed us to test multiple possible objective functions in our search for a simple but realistic function (§3.3.2). We provide the details of our Genetic Algorithm in the supplied Matlab code [68] and the following section.

3.4 Details of the Genetic Algorithm

In this section we provide some of the details of the Genetic Algorithm.

Inputs

- Matrix containing the coordinates and population of each *Point of Presence* (PoP).
- The optimization parameters: $k_0, \ldots, k_3$.
- The genetic algorithm settings, including the number of chromosomes in a generation and the number of generations. More settings appear below, in italics.

Outputs

- Adjacency matrix of the best topology found by the Genetic Algorithm.
- The routing matrix for the best topology found by the Genetic Algorithm.
- Link capacities for the best topology found by the Genetic Algorithm

- (Optionally) Adjacency matrices, routing matrices and link capacities of the whole population in the final generation.
- Costs of the candidate topologies in the final generation.

State

- Each candidate topology in the current generation is stored as an n by n adjacency matrix.
- The costs for each topology are also stored.

3.4.1 Algorithm

1. Determine the first generation of topologies
   - One starting topology is the minimum spanning tree (using the physical distances determined by the PoP-positions in the input).
   - One starting topology is the fully connected topology (every PoP is linked directly to every other PoP).
   - Topologies can be provided directly as input, typically from other optimization methods.
   - The remaining topologies are generated randomly using Erdos-Renyi graphs with a chosen probability for each link. This probability p can be fixed over all these topologies, or be different for each topology as desired. We use a value of p such that $p\binom{n}{2}$ is approximately equal to the expected number of links in the optimal topology; approximations to the optimal number of links can be obtained through previous runs of the algorithm. This aids convergence speed of the genetic algorithm but is otherwise unnecessary.
2. Evaluate the cost of each of the topologies in the current generation.
3. Create the next generation of topologies. These consist of:
   - The best *num_saved_topologies* topologies from the previous generation.
   - *num_crossover_topologies* topologies resulting from crossover (breeding).
   - *num_mutation_topologies* topologies resulting from mutation.
4. Repeat from Step 2 until there have been *num_generations* generations.
5. Output the topology with the lowest cost.

For a Genetic Algorithm to be effective, it must be possible to efficiently generate "better" topologies by breeding and mutating "good" topologies. Consequently, the key challenge in designing a Genetic Algorithm is designing the crossover and mutation steps so that they work quickly and have a reasonable likelihood of producing good topologies.

Crossover

Crossover involves choosing several topologies ("parents") from the current generation to combine and create a new topology of the next generation. COLD picks b topologies uniformly at random as candidates to become parents, then chooses the best a of them as parents for a crossover. This process occurs once for each new topology created by crossover. We typically chose $a = 2$ and $b = 10$. Choosing parents this way ensures that the worst topologies will not become parents, and it induces a strong bias towards the better topologies as parents. For instance, with $a = 2$ and $b = 10$ there's only a 10% chance to have any of the worst 50% of topologies as parents in any given crossover event. We chose $a = 2$ and $b = 10$ since it seemed to produce a good tradeoff between convergence speed (number of generations until the best cost was not changing very often) and reliability (that is, finding similarly costed topologies over several runs of the GA). If smaller values of b and larger values of a were used, this would allow more variety to be kept at each generation, at the expense of the average cost of topologies across that generation.

Once the parents of the new topology are chosen, it simply remains to generate the new topology from them. Since each topology is a graph with n nodes, there are $\binom{n}{2}$ possible links in the new topology. For each of these possible links, we choose one of the a parents at random and copy whether the link exists or not from that parent. When choosing the parents at random, they are chosen with probability inversely proportional to their cost. This crossover step occurs once for each of the new chromosomes created by crossover.

Mutation

To create a mutated topology, one of the topologies from the previous generation is selected at random, with probabilities inversely proportional to cost. Then one of two types of mutation occurs:

- *Link mutation*: A pair (m_+, m_-) is determined using a random function *mutate_fn()*. m_+ links that exist in the chromosome are removed, and m_- of the links that do not exist are added to the chromosome. We choose mutate_fn() so

that m_+ and m_- are both geometric random variables with parameter 0.5, giving an average of two link changes each time a mutation occurs.

- *Node mutation*: One of the non-leaf nodes is chosen uniformly at random and made into a leaf node, with its only link now running to the closest non-leaf node.

Connectedness

The mutation and crossover steps can produce a network that is disconnected. If this occurs, COLD finds all the connected components and the shortest link between each pair of connected components. COLD then finds a minimum spanning tree (minimum in terms of physical link distance) to connect these components. This ensures that the resulting networks are always connected. Typically it is used very rarely. However, when the costs induce topologies with low numbers of links, this step becomes much more frequent, although it is still rare to be forced to add more than one or two links to ensure connectedness.

3.5 Performance of the GA

The first issue to consider is the tuning of the internal parameters of the GA to produce near-optimal topologies while managing its run-time. These internal parameters include the number of generations, the number of topologies in each generation, and the mutation and crossover parameters. To compare the effectiveness of different parameters we could simply run the GA with the fixed input and context, varying each internal parameter. The best internal parameters would be those producing the lowest cost topologies at a reasonable speed. However, we would still have no idea whether these topologies are near the optimum.

The most obvious approach to testing if topologies are optimal is to evaluate every possible graph. While exhaustive enumeration is totally infeasible for large numbers of nodes, we can still use it on contexts with low numbers of PoPs to find the optimum topology. We tested on a wide variety of contexts of up to 8 PoPs and the tuned Genetic Algorithm found the optimum in every case. The number of generations and the number of PoPs in each generation needs to grow as the number of PoPs in the network grows, but this testing determined an absolute minimum for these parameters. To reliably find the optimal network with 8 nodes, we required a population size of about 20 and 20 generations.

Brute force testing is not sufficient to demonstrate the performance of the GA for the full range of network sizes. We implemented several algorithms to test against the GA

for higher numbers of PoPs. The number of possible graphs is super-exponential in the number of nodes, so these algorithms reduce the size of the problem by focusing on hub nodes and their interconnections. To further reduce the magnitude of the problem we used greedy algorithms.

Each test algorithm starts with one hub node, and every other node a leaf connected to the hub. Hubs are then added and connected to the closest leaf nodes (the way in which the hubs connect to each other varies) in such a way that the cost of the network reduces with each hub added. If a hub can not be added without increasing the cost of the network, the algorithm terminates. The alternate methods for adding hubs are as follows:

- *Random Greedy*: A random permutation of all the nodes is chosen. The algorithm then iterates over the PoPs in this order. For each PoP it decides whether changing it to a hub reduces the cost of the network, and if so, the node is added as a hub. New hubs are linked to the existing hubs in a greedy manner: picking the best connecting link (the one that gives the lowest cost network), then the next best link, etc., until there are no more cost reductions. Once all the PoPs in the permutation have been evaluated, the process repeats for many different random permutations of the PoPs. The best network at the end is chosen.
- *Complete*: All the PoPs are tested as a possible hub and the best one is taken. This repeats until none of the remaining nodes will reduce the cost when added as a hub. Each new hub is connected to all the existing hubs, thus making a network where the hubs form a completely connected subgraph (a clique).
- *MST*: Just like complete, but the hubs are connected in a minimum spanning tree.
- *Greedy attachment*: Like complete and MST, but inter-hub connections are chosen greedily for each new hub (as in Random Greedy).

After tuning the GA we compared it to each of the greedy algorithms by generating a set of contexts and running each algorithm on each context. We then compared the costs of the best solutions found by each algorithm. The cost of the optimal solution found by each algorithm is plotted against k_2 in Figure 3.1. It is clear that different algorithms perform better in different circumstances (with different values of k_2, k_3). With low k_2 inducing low numbers of links and higher k_3 inducing a low number of hubs the Random Greedy algorithm performs poorly, but at most 25% worse than the best of the other algorithms. Greedy attachment appears to work well uniformly, but its runtime grows very quickly in the number of nodes (greater than $O(n^4)$), so it rapidly becomes impractical for higher numbers of nodes (it is much slower than the GA even for $n = 50$).

Note that when the greedy algorithms perform better than the GA, they can be run first and the results used as starting topologies for the GA (this is labeled as *initialized GA* in Figure 3.1) ensuring it provides topologies that are at least as good as any produced

by the greedy algorithms. The greedy algorithms are fast for moderate numbers of nodes, thus running them before the GA to improve the overall results is worthwhile.

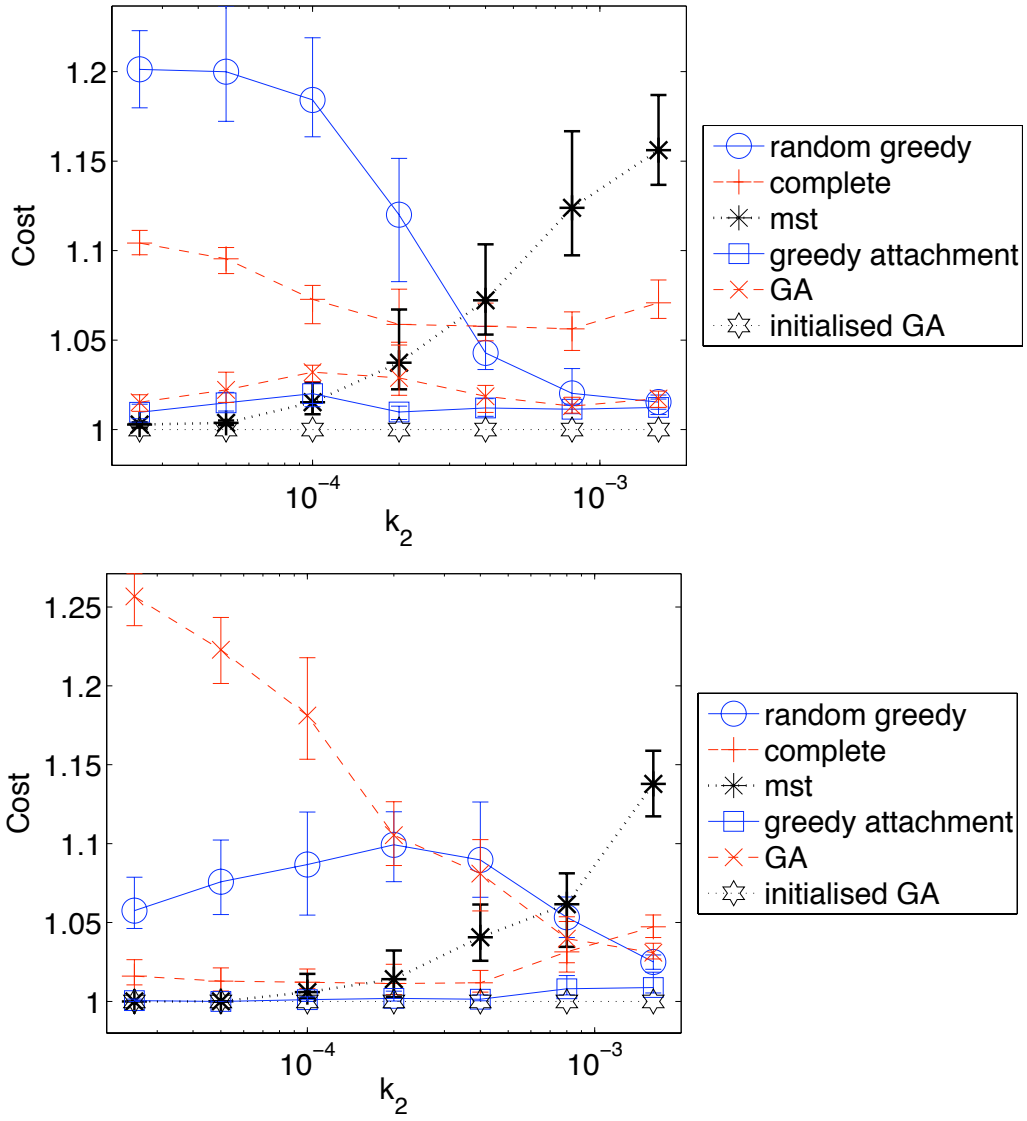


Figure 3.1: Cost of best solution versus k_2, normalized by the initialized GA result, $n = 30$. Error bars denote 95% bootstrap confidence intervals for the mean of the results, 20 trials for each set of parameters. $k_3 = 0$ *(top)* and $k_3 = 10$ *(bottom)*. In both figures, $k_0 = 10, k_1 = 1$.

Figure 3.2 shows the runtime of the GA, which grows as $O(n^3GT)$, where n is the number of PoPs, G is the number of generations and T is the number of new topologies in each generation. We chose to fix G and T at 100 in the tuning stage. The n^3 term arises in evaluating the all-pairs shortest paths step, though this could be performed faster with a better implementation.

$O(n^3)$ might seem impractical as an order of scaling for the run time of the GA, but we generated networks of up to 800 PoPs with it (very large for a PoP-level network). As an example, it takes more than an hour to generate each network of this size on a 2009

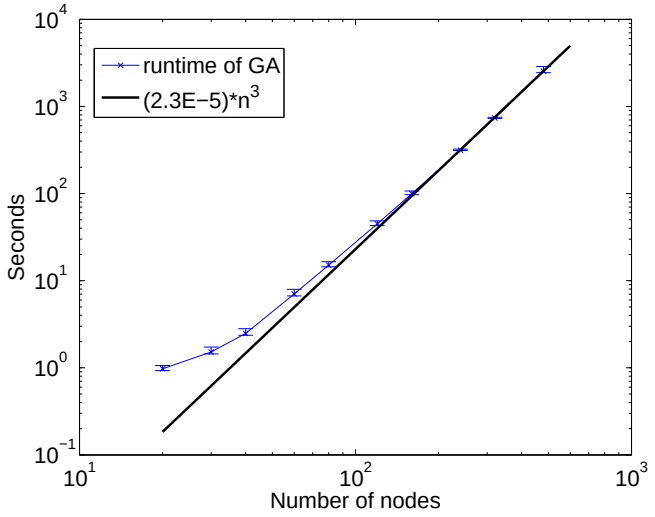


Figure 3.2: The run time of the genetic algorithm grows cubically in the number of nodes, with $G = T = 100$.

Macbook Pro. This reinforces our decision to optimize at the PoP-level as router-level optimization could produce problems of totally infeasible size.

As further validation, the GA was tested for sensitivity to increasing the number of generations or the number of topologies in its population of solutions. Despite quadrupling the number of topologies and the number of generations, the GA showed at most a 10% decrease in cost for all values of (k_0, k_1, k_2, k_3) tested. The number of generations and size of population chosen represent a reasonable trade-off between speed and optimality.

3.6 Tunability

One of our goals is that the output of the optimization process be tunable, as expressed in the introduction. Here we show the simple relationship between the input parameters k_0, k_1, k_2, k_3 and some regularly observed characteristics of the output networks. We can finely control such statistics as average node degree, assortativity of node degrees, clustering coefficient, average shortest path lengths, average node and link betweenness by changing these parameters. We can also control them such that they are representative of any range of networks found in [3].

COLD has four control parameters k_0, k_1, k_2, k_3 that we can tune to produce different characteristics in the output topologies. However, if we multiply all of the parameters by a constant, the costs for every topology will be multiplied by that constant, resulting in an equivalent optimization problem. Consequently, we fix k_0. When tuning these parameters we noticed that increasing k_0 or increasing k_1 had very similar effects: they

both serve to reduce the number of links in the topologies, and produce very similar effects in the measured statistics. We therefore kept the ratio between them fixed, with $k_0 = 10$ and $k_1 = 1$.

We show some of the most pertinent statistics for the synthesized topologies, and how they relate to k_2 and k_3. Each of these plots is continuous and monotonic with tight error bars, so for any given statistic it is simple to read off some values of k_2 and k_3 that produce the desired value of that statistic. This makes it easy to tune COLD to synthesize networks with the desired value for any of these statistics. Further, the values obtainable for each statistic covers the full range of the values realized in [3], see Figures 3.3, 3.4 and 3.5. Here we show only the graphs for $n = 30$, as they are similar for other values of n, including $n = 50$ and $n = 80$. We compare the statistics obtainable by varying k_2 and k_3 with those for networks of similar size in [3] (between 27 and 33 nodes, inclusive).

The average node degree is a frequently used statistic, representing how many links there are in the network. The higher the node degree, the more "meshy" the graph is. Since the costs associated with k_0, k_1 (and to a lesser extent k_3) only increase when links are added to the network, increasing k_0, k_1, k_3 will reduce the average node degree. Increasing k_2 instead increases the average node degree, as is shown in Figure 3.3. Note that for very high values of k_3 the networks are all of the same form: one hub PoP connected to 29 leaf PoPs. These networks are trees so they have the minimum possible node degree for 30 nodes, $2 - \frac{1}{30}$. While not shown in Figure 3.3, by increasing k_2 further we can increase the average node degree to the maximum possible node degree of 29. When the value of k_2 strongly dominates, it produces a fully connected graph, confirming our theoretical assertions in §3.3.2.

The majority of PoP-level networks in [3] display average node degree ranging from the minimum available, up to around 4.5 (Figure 3.3). COLD can produce average node degree values all the way from the maximum of $n - 1$ to the minimum of $2 - \frac{1}{n}$ for any value of n.

The diameter of a graph is another frequently used statistic [32]. It denotes the maximum number of hops between pairs of nodes in the graph. Graphs with small diameter relative to their size and node degree demonstrate the "small-world" property. Graphs synthesized by COLD can be controlled to display a wide variety of diameters. The 30 node graphs in Figure 3.4 display a wide range of diameters. To obtain very large diameters, the number of links in the graph must be low (low k_2) but also the hub cost k_3 must be very low. A high hub cost forces a low number of hubs, causing the diameter in the hub subgraph to be low, and hence the diameter in the overall graph to also be low (since every leaf is directly connected to a hub). 90% of the appropriately sized networks from [3] have diameters of 13 or less. One network has a diameter of 16, a very large diameter for a network with 30 nodes. Upon inspection, this graph spans

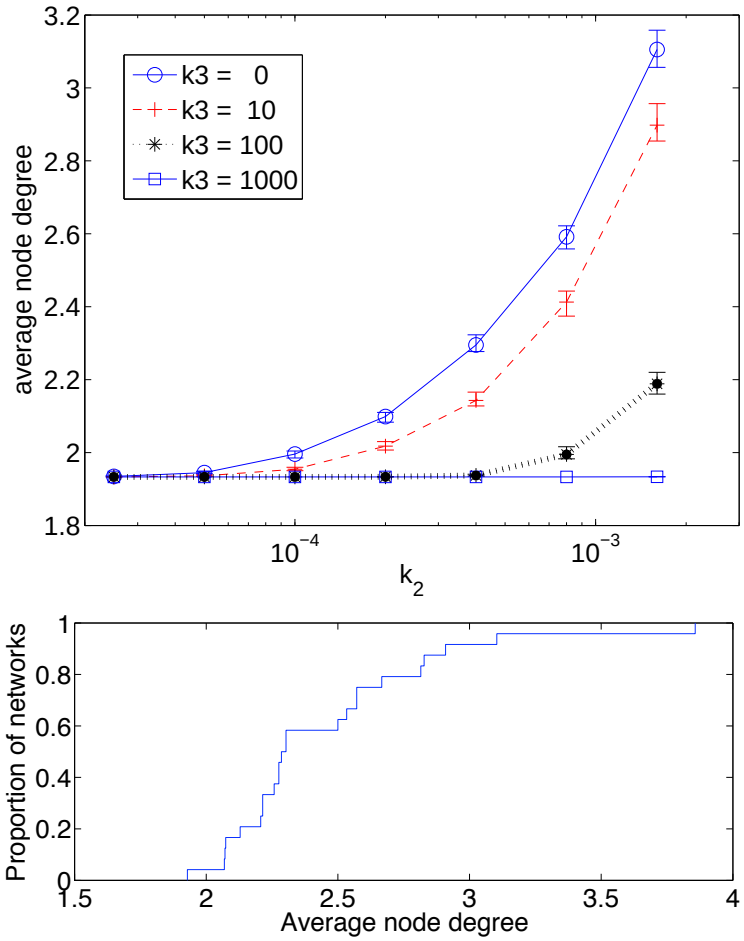


Figure 3.3: *(top)* The average node degree in the solution versus the value of k_2, for various values of k_3, with $k_0 = 100$, $k_1 = 7$. *(bottom)* Empirical distribution function for average node degree of networks in [3].

the islands of Java and Bali in Indonesia – a very long, thin region. Since the COLD graphs we are comparing it to are generated on square regions, they are very unlikely to have this diameter.

Clustering coefficients are a way of measuring locality, a principle commonly referred to in network design. The global clustering coefficient (GCC) measures the number of triangles present in the graph compared to the maximum number of triangles possible. In [3] 90% of the GCCs are below 0.25, and all of the higher GCCs belong to networks with very few nodes. Varying the value of k_2 causes COLD to move from producing trees (GCC of 0) to producing fully connected graphs (GCC of 1), importantly the GCC is controlled finely and reliably by k_2 and k_3, allowing the degree of locality present in the synthetic graphs to be finely tuned across all possible values, as shown in Figure 3.5.

We also provide here some output topologies to give tangible examples of what the output of COLD looks like for some standard sets of parameters (Figures 3.6 and 3.7). We show the affect of increasing k_2 while keeping the other parameters constant – a

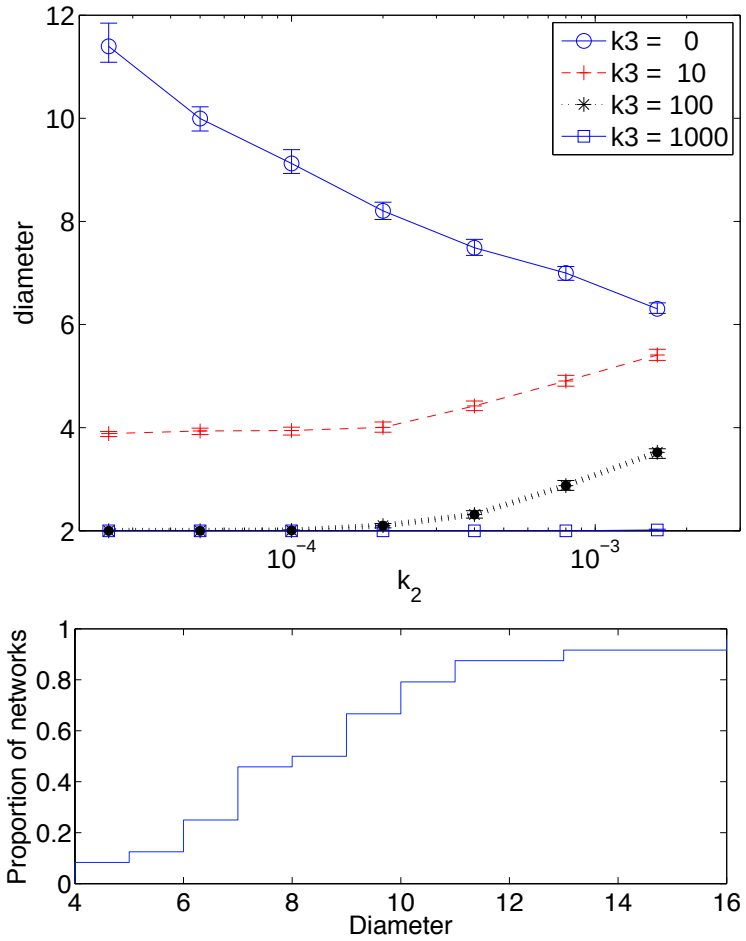


Figure 3.4: *(top)* Attainable diameters for differing values of k_2 and k_3, with 30 node synthesized networks. *(bottom)* Distribution of diameter in networks from [3].

noticeable change in the node degree and an increase in the number of hubs.

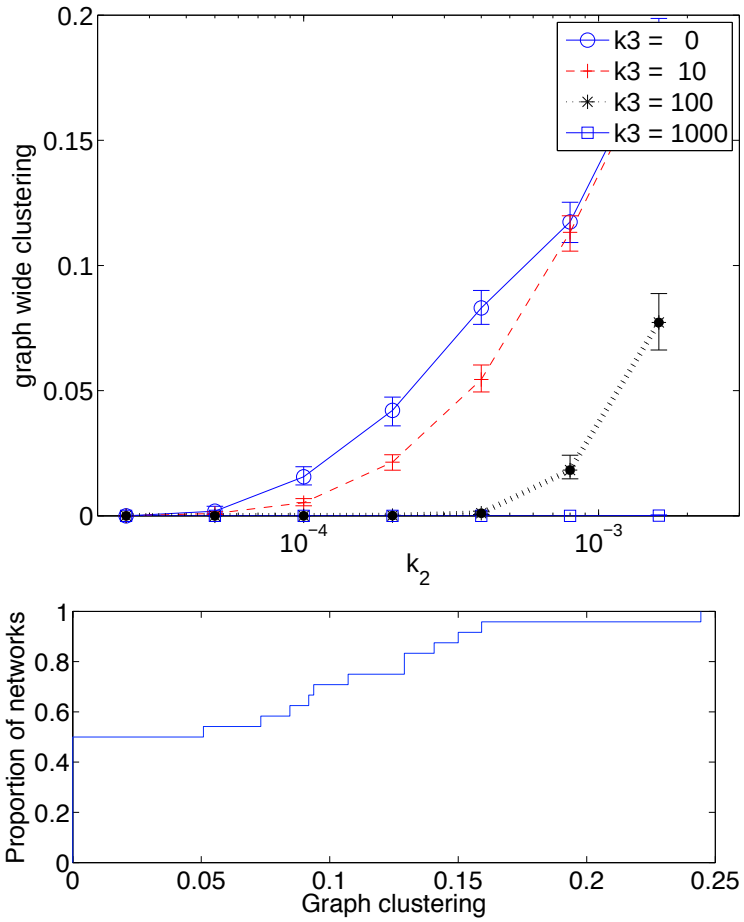


Figure 3.5: *(top)* Global clustering coefficient. *(bottom)* Distribution function of global clustering coefficient for networks in [3].

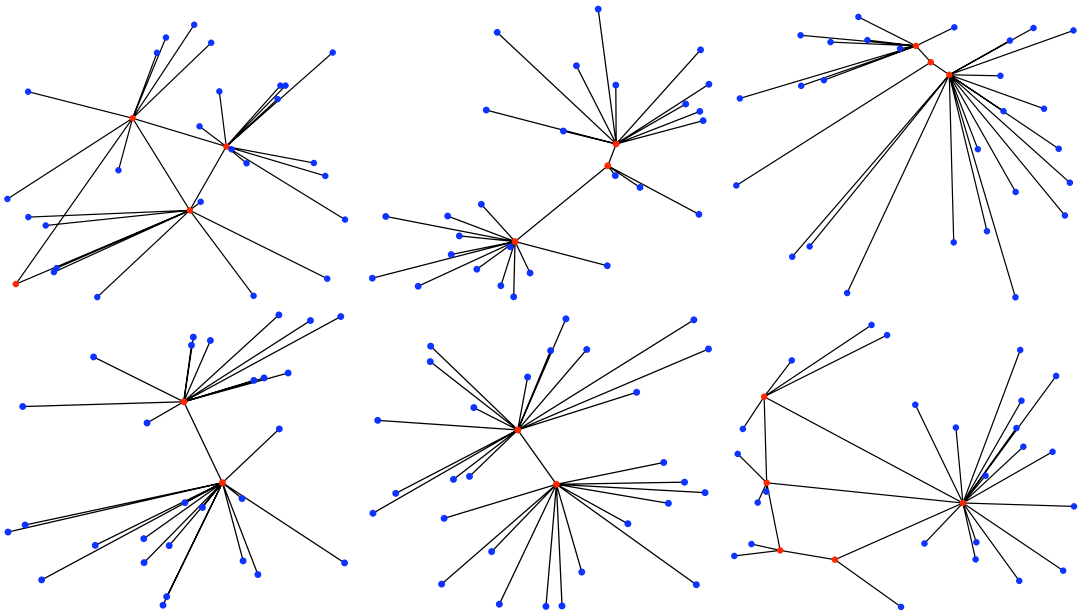


Figure 3.6: Example topologies when $[k_0, k_1, k_2, k_3] = [10, 1, 6 \times 10^{-4}, 20]$. Hubs are in red and leaves are in blue. Generated using example2.m from [68]. Here the hub cost (k_3 cost) is relatively high and the bandwidth cost (k_2 cost) is low so we see sparse networks with few hubs.

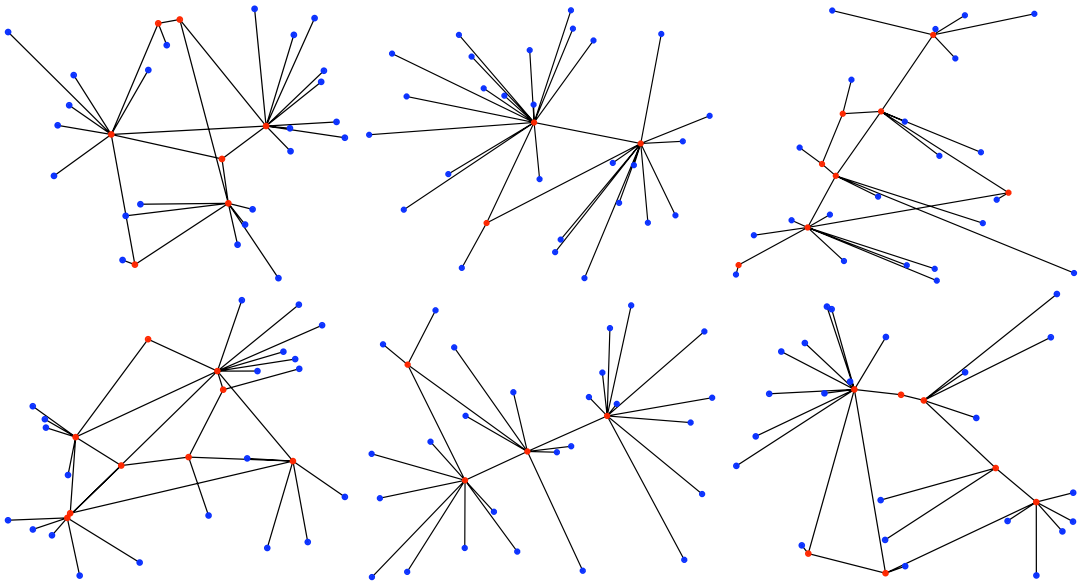


Figure 3.7: Example topologies when $[k_0, k_1, k_2, k_3] = [10, 1, 10^{-3}, 20]$. In comparison to Figure 3.6, the k_2 cost is higher and so we see more links and more hubs.

3.7 Leaf-inducing cost

In this section we show that while it is possible to generate reasonable networks just using link costs $\sum_{i \in E}(k_0 + k_1\ell_i + k_2\ell_i w_i)$, a node-based cost $\sum_{j \in N_C} k_3$ is required to encompass all the variety we see in networks in [3].

When generating networks without a node-based cost, we can still vary k_2 to control the statistics mentioned above to obtain any value realized by the networks in [3], including diameter, average node degree, clustering, and others. However, there are some networks in [3] that have very many PoPs of degree 1 (leaf PoPs), and few higher degree PoPs (hubs). One such example is in Figure 3.8. This "hubbiness" is reflected in the coefficient of variation of node degree (CVND). CVND is defined as the standard deviation of the node degrees divided by the mean of the node degrees. Some networks in [3] have a CVND of nearly 2, whereas with $k_3 = 0$ none of the synthetic networks have a CVND greater than 1.

Before introducing a node cost we tested if it was possible to generate hubbier networks (with higher CVNDs) by changing the nature of the context input to the optimization. There are two parts of the context that can be altered:

- The populations of nodes.
- The spatial distribution of the nodes.

Earlier, the node populations were exponentially distributed with fixed mean. To get a greater variation in node degree, we trialled a heavy-tailed distribution, one more likely to produce extreme values. In this case, we used Pareto distributions (a form of power-law distribution) with the same mean as the exponential. The Pareto distribution has two parameters: a shape parameter α and a scale parameter x_m. The lower α is, the more heavy tailed the distribution. In this case, the Pareto distributions used had $(\alpha, x_m) = (1.5, 10)$ and $(1.11, 3)$. Both choices have the same mean as the exponential case, but infinite variance.

It is also possible to change the spatial distribution of the nodes. Earlier, the nodes were independently uniformly distributed over a rectangle. In order to get a few nodes with very high degree and many nodes with low degree we tried clustering the nodes together by randomly choosing the locations of a few small circles, and restricting nodes to lie within these circles. We also tested the effects of changing their number and size.

Both the changes to the population distribution and the spatial distribution successfully increased the CVND and increased the number of leaves. However, even at their most extreme they did not reduce the number of hubs sufficiently to represent the full gamut of networks – it was necessary to add a hub cost to allow for the sort of variability we see in [3]. Once a hub cost is incorporated, it is possible to accurately control both

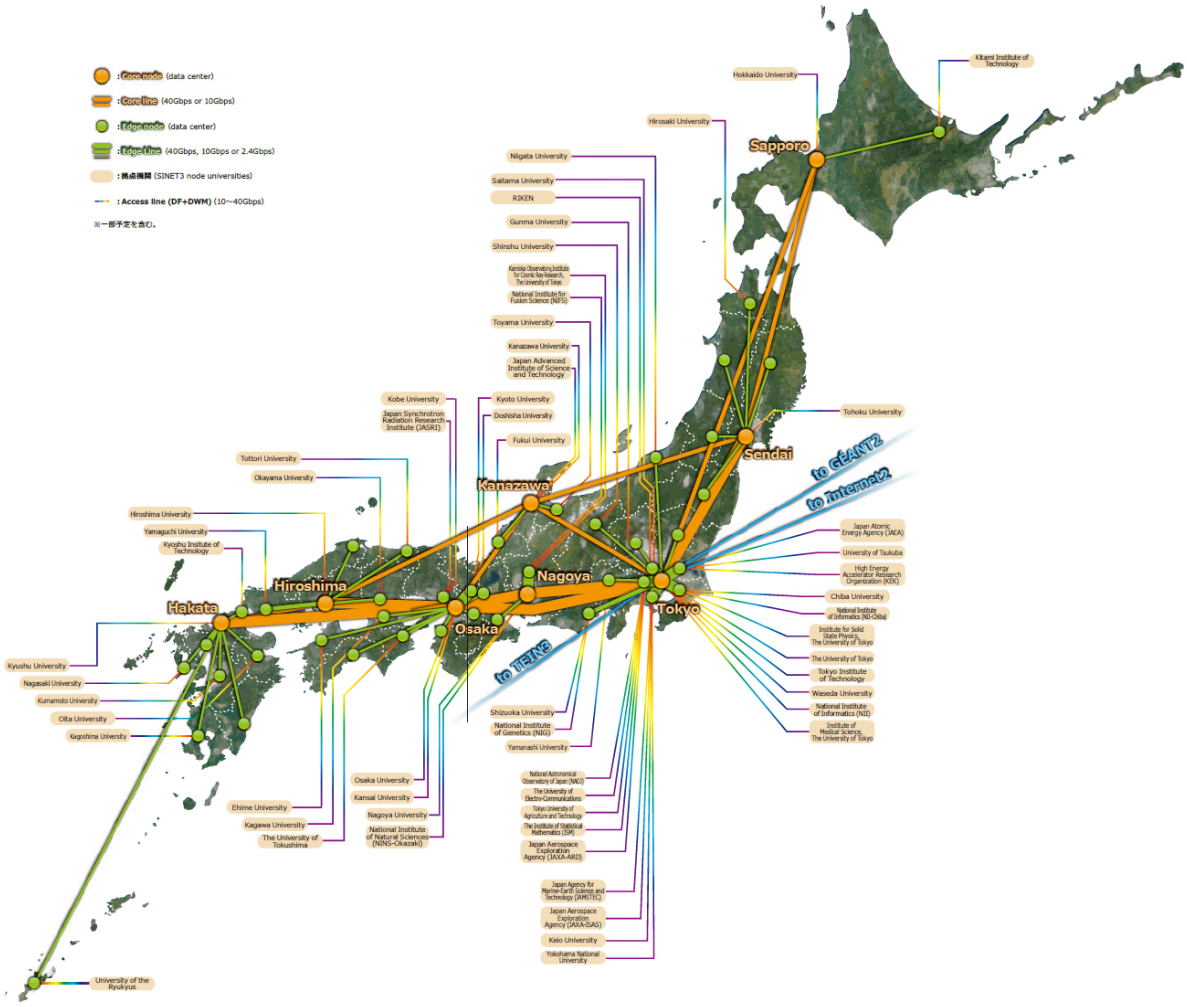


Figure 3.8: An example network from the Internet Topology Zoo [3] with high coefficient of variation of node degree. Note that some PoPs are designated as core PoPs, and some are designated as leaf PoPs. Core PoPs are those that are directly connected to more than one other PoP, just as in our optimization formulation.

the CVND and the number of hubs (Figures 3.9 and 3.10) even with exponentially-distributed population and uniformly-distributed locations. Increasing k_3 allows the CVND to exceed 2.5 for networks with 30 nodes (and higher levels for higher numbers of nodes). Unsurprisingly, the value of k_2 also matters, since high values of k_2 tend to induce networks with more links, and more direct links, increasing the number of hubs. High values of k_3 relative to k_2 are easily sufficient to cover the variation of CVND we see in [3] (Figure 3.9). Similarly, high values of k_3 relative to k_2 are capable of reducing the number of hubs down to 1, the minimum possible (Figure 3.10). Note also that the maximum number of hubs is reached when k_2 becomes sufficiently high relative to k_0, k_1 and k_3, since in that case the networks produced are cliques.

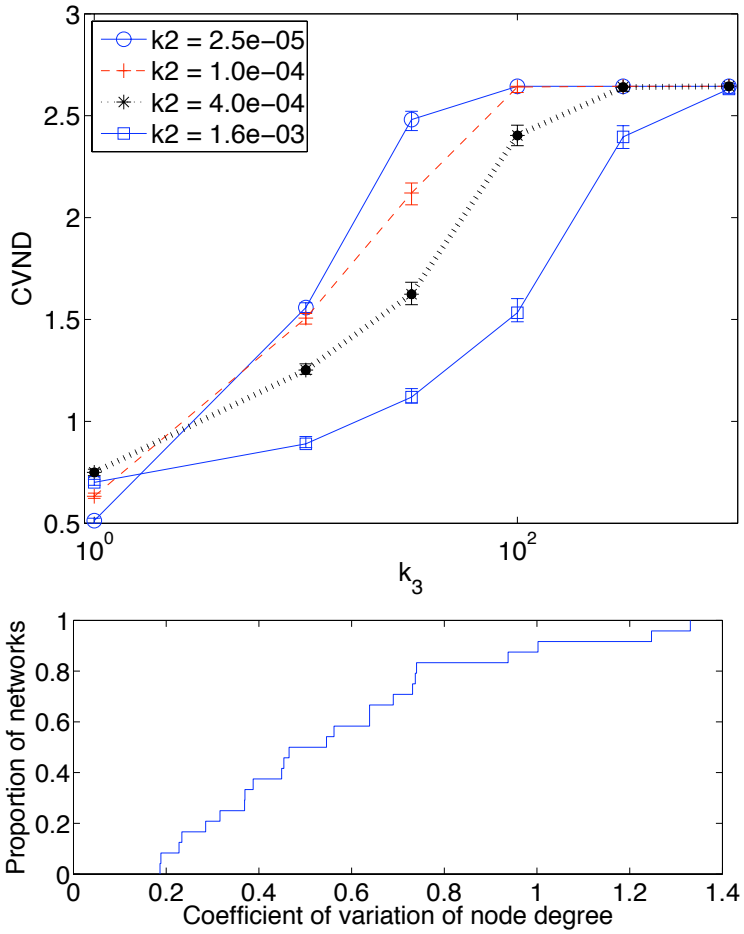


Figure 3.9: *(top)* Coefficient of variation of node degree versus k_3, for differing values of k_2. Without a high value of k_3, it is extremely rare to produce a CVND over 1. *(bottom)* Distribution of CVND degree for networks in the [3]. About 15% of the networks have a CVND over 1, a value unattainable without a node-based cost.

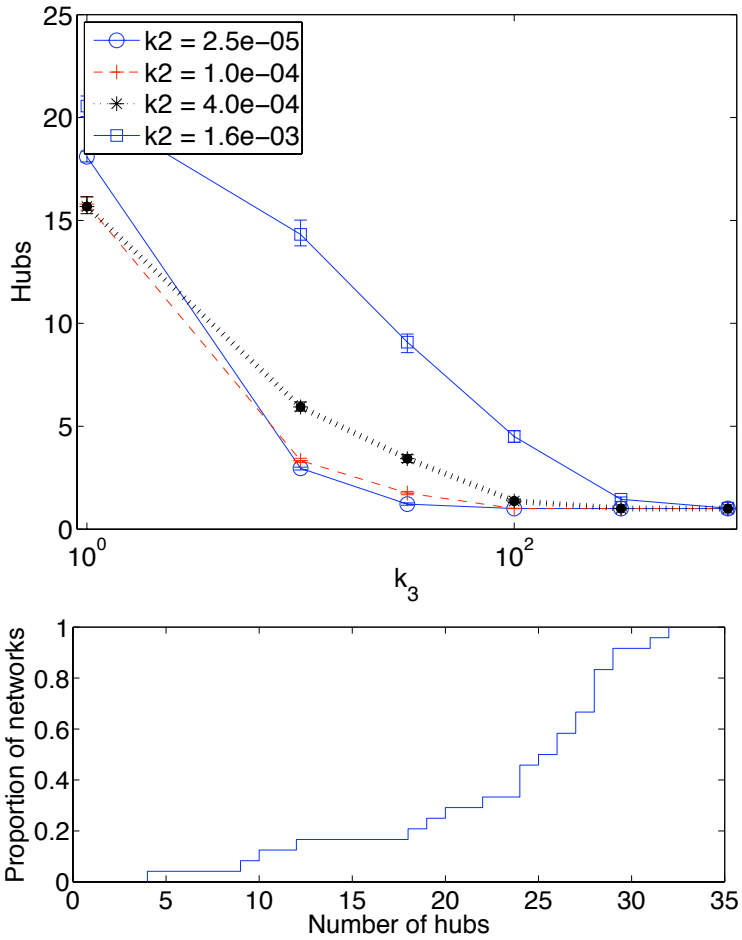


Figure 3.10: Number of core PoPs, for networks with 30 nodes. Plotted against k_3 for various values of k_2. This plot shows that low numbers of core PoPs (high numbers of leaves) are achievable by using high values of k_3. While many networks in the zoo have greater than 21 nodes, this is easily achieved for higher values of k_2 or lower values of k_3(not shown in figure).

3.8 From PoP-level to router-level

So far we have been concerned with synthesizing PoP-level topologies. These are useful for many tasks, but often router-level topologies are needed. In this section we present a method to take a PoP-level topology as input and synthesize a router level topology. The input PoP-level topology can be supplied by the process described above, some other method, or even a measured PoP-level topology [58,69]. If the PoP-level topology comes with other meta-data like traffic demands they can be used to inform the choices made when determining the router-level topology. This is a great advantage to using the COLD genetic algorithm for PoP-synthesis – it comes with additional context that helps to determine the structure of the router level network. For instance, PoPs that serve a larger number of providers require more routers with larger bandwidth.

Optimizing a network at the router-level is sometimes possible, but in practice it is rarely attempted as the problem is often much larger and more complex than at the PoP-level (*e.g.,* a moderately sized network may have 40 PoPs, but 200 routers). There are also additional technical and engineering constraints at the router level, such as the complex interaction between line card options and the number of router ports and the requirement for redundancy in parts of the network. The combination of these factors makes strict optimization at this level prohibitively complex. Further, a network resulting from such an optimization can be very complex and difficult to manage or debug effectively. Instead of optimizing at the router level, COLD is inspired by a standard approach used by network designers. COLD uses a hierarchical design methodology and templated design to both operate more efficiently and synthesize more realistic network topologies.

Network operators leverage hierarchy [54–56] in network designs for two main reasons:

- *Scalability:* in many cases a hierarchical network can be made much larger than the alternatives.
- *Simplicity:* hierarchy makes a network easier to manage. It is analogous to modularity in programming languages — ideally it allows differing levels of the hierarchy to be considered in isolation.

Most frequently a network's hierarchy is based on the natural structure of its PoPs, which roughly correspond to a network's ability to provide service to a metropolitan area. A common design strategy is to heuristically optimize the PoP-level network, and then introduce hierarchy by using templated design for the PoPs [55, 56, 67]. Using a small number of templates (repeated patterns) has many advantages: there are only a few designs for engineers to learn and debug, the amount of documentation needed is reduced, spare inventory can be reduced (as fewer models of device are needed), and devices can be purchased in larger quantities, reducing cost. Ideally all PoPs would use

the same pattern, but in reality some are many times the size (in terms of customers numbers or traffic) of others and one size does not fit all. Nonetheless, a small set of designs often suffices.

Previous work has used PoP structure in the inference of network topology [58,69], and in network design [57]. Here we have more information because we start from a *context*, which provides geographical and traffic information. COLD applies an approach motivated by the work of Parsonage *et al.* [57], who show that many real world networks can be described as the graph product of the PoP-level graph and a few PoP design templates. The reason for this is clear: network designers use templates to design PoPs [54–56] and features such as link redundancy are naturally expressed in terms of the product used. This correspondence between the mathematical concept of a graph product and the choices made by network designers enables us to explicitly build a network using the same constraints as real network designers. This meets our requirement that a network's form and structure be driven by technical constraints.

Before we can use graph products to synthesize router-level networks, we have to define what we mean by graph product. We use $\mathcal{G}$ to represent a graph, and define $N(\mathcal{G})$ to be the nodes or vertices of $\mathcal{G}$ and $E(\mathcal{G})$ to be the edges. Graph products are a standard notion in graph theory, that of a product between two graphs $\mathcal{G}$ and $\mathcal{H}$ that produces a third graph $\mathcal{G} \times \mathcal{H}$. The node set of $\mathcal{G} \times \mathcal{H}$ is just the cartesian product of the nodes in $\mathcal{G}$ and $\mathcal{H}$,

$$N(\mathcal{G} \times \mathcal{H}) = N(\mathcal{G}) \times N(\mathcal{H}).$$

Consequently we write nodes in $\mathcal{G} \times \mathcal{H}$ as an ordered pair (u, v) with $u \in N(\mathcal{G})$ and $v \in \mathcal{H}$. While the *node* set is usually defined this way, there are a wide variety of "correct" ways to define the *links* in the graph product. We can define the different products by considering the connectivity between two arbitrary vertices (u, v) and $(u', v') \in N(\mathcal{G} \times \mathcal{H})$. Some examples of graph products are:

- The *Cartesian product* $\mathcal{G} \square \mathcal{H}$: A link exists between two nodes (u, v) and (u', v') iff $u = u'$ and $(v, v') \in E(\mathcal{H})$ or if $v = v'$ and $(u, u') \in \mathcal{G}$.
- The *Tensor product* $\mathcal{G} \otimes \mathcal{H}$: A link exists between two nodes (u, v) and (u', v') iff $u \neq u', v \neq v', (u, u') \in \mathcal{G}$ and $(v, v') \in \mathcal{H}$. The Tensor product is also known as the *direct*, *relational*, *categorical* or *cardinal* or *Kronecker* product.
- The *Strong product* $\mathcal{G} \boxtimes \mathcal{H}$: Links in $\mathcal{G} \boxtimes \mathcal{H}$ are a union of those in $\mathcal{G} \square \mathcal{H}$ and $\mathcal{G} \otimes \mathcal{H}$.
- The *Lexicographic product* $\mathcal{G} \bullet \mathcal{H}$: Two nodes (u, v) and (u', v') are adjacent iff $(u, u') \in E(\mathcal{G})$ or if $u = u'$ and $(v, v') \in E(\mathcal{H})$.
- The *Rooted product* $\mathcal{G} \circ \mathcal{H}$ requires specification of a *root node* $h \in N(\mathcal{H})$. Then

(u, v) and (u', v') are adjacent iff $v = v' = h$ and $(u, u') \in E(\mathcal{G})$ or if $u = u'$ and $(v, v') \in E(\mathcal{G})$.

We can view these graph products (except the Tensor product) as making $|N(\mathcal{G})|$ copies of the graph $\mathcal{H}$ (one corresponding to each node of $\mathcal{G}$), then connecting them according to rules that depend on which graph product is being used. We call each copy of $\mathcal{H}$ an $\mathcal{H}$-fiber. Note that in the Tensor product, the group of vertices corresponding to a specific node in $\mathcal{G}$ do not form a copy of the graph, unlike the other graph products listed here.

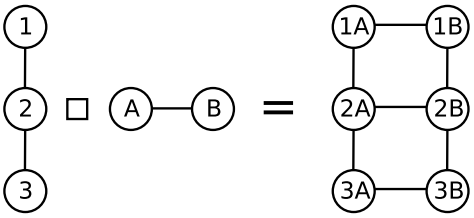


Figure 3.11: Cartesian product.

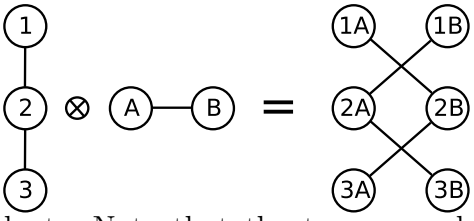


Figure 3.12: Tensor product. Note that the tensor product can produce a graph with more than one connected component (a disconnected graph) from two connected graphs. This is partly because the $\mathcal{H}$-fibers in $\mathcal{G} \otimes \mathcal{H}$ never contain any links.

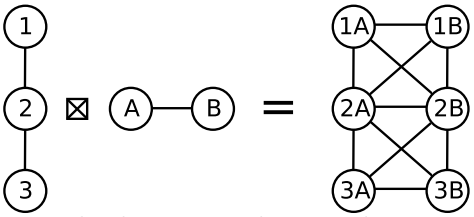


Figure 3.13: Strong product. The lexicographic product produces the same output as the strong product in this case (when $\mathcal{H}$ is a clique).

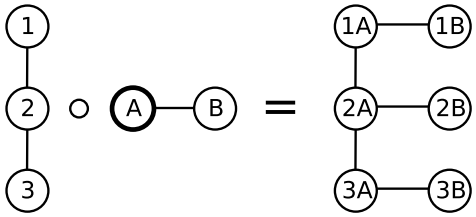


Figure 3.14: Rooted product, with A as the root node in $\mathcal{H}$.

We can use one of these standard graph products for producing a router-level topology as follows:

1. Call the PoP-level topology $\mathcal{G}$.
2. Choose a topology as a template for the PoPs. Each PoP will this topology as its internal structure $\mathcal{H}$.
3. Choose a level of connectivity between PoPs – this determines the graph product used. To get the most basic connectivity with no redundancy, choose Rooted product. To get a higher level of redundancy use Cartesian product or even Strong product.
4. Form the chosen graph product $\mathcal{G} \times \mathcal{H}$. Each $\mathcal{H}$-fiber represents the routers forming one PoP. Two PoPs are directly connected with router level links iff there is a connection between them in the PoP graph. The strength of connections between PoPs is determined by the type of graph product chosen.

This gives us a basic way of producing a highly structured network with strong repeated patterns using basic graph products. However, the simple standard graph products from the graph theory literature are not sufficient for our network design needs. Using basic graph products results in the same structure in every PoP, and the same form of connection between each pair of connected PoPs. Instead, for COLD we choose to use *generalized graph products*, as detailed in [57]. These allow us to have differing intra-PoP designs, and differing levels of connectivity between adjacent PoPs. Generalized graph products allow the $\mathcal{H}$-fiber to depend upon labels on the nodes of graph $\mathcal{G}$ (the PoP-level graph). They also allow each pair of $\mathcal{H}$-fibers to separately choose the type of product used, be it Rooted product, Cartesian product, Strong product, etc.

Many methods of synthesis are possible using the techniques in [57], allowing great variety in the design rules to be applied. In keeping with the requirement for simplicity we choose a small subset here:

- Generally we use M PoP *templates* — small graphs that describe the core structure of the PoPs. In order to illustrate the method we choose $M = 2$ and the simplest

possible templates, a single node, or a pair of connected nodes, allowing us to express a requirement for node redundancy in a PoP.
- We restrict ourselves to using the Cartesian product and the Strong product.

In the language of [57], we write the graph product that generates the core-router-level network as

$$\mathcal{G} \otimes \{\mathcal{H}_i\},$$

where $\mathcal{G}$ gives the PoP-level topology, and the $\mathcal{H}_i$ are chosen from the template graphs $\mathcal{H}_a = (\mathcal{N}_a, \mathcal{E}_a) = (\{1\}, \phi)$ and $\mathcal{H}_b = (\{1, 2\}, \{(1, 2)\})$.

The different templates and products are assigned in a way that satisfies a pair of simple design principles:

- more important PoPs and links carry more traffic;
- more important PoPs and links require redundancy.

We apply these principles by selecting two sets of thresholds: one with $M-1$ node-traffic thresholds for selecting the PoP templates to be used and one with $K-1$ link-traffic thresholds for selecting the graph product (level of redundancy) to be used. Thus, in our example there is one node-traffic threshold and one link-traffic threshold. When the total traffic through a PoP exceeds the node-traffic threshold we assign it a two-router PoP otherwise a one-router PoP. Likewise, if a link carries more than the link-traffic threshold it is assigned the strong product, and otherwise the Cartesian product.

The thresholds are new parameters, however, in keeping with our guiding principles they are simple, intuitive and meaningful. It is easy to design a network with both node and link redundancy, simply set the thresholds to zero. Or we could assign the node-traffic threshold such that leaf-PoPs are assigned one router, and all others two. Or we could choose the thresholds to be large to ensure that the generated networks have little redundancy. We could even start from the same PoP-level graph, and generate a range of networks with different levels of redundancy by varying the thresholds.

The elegant thing about the graph-product approach is that we can naturally use the link capacities, or links weights of the inter-PoP graph, in conjunction with the template graphs $\mathcal{H}_i$ (which can contain labels such as router types or models) to generate the capacities and link weights needed in the final graph product.

The final step of the process is to introduce one more layer of hierarchy – the above generates core or backbone routers. We also determine the correct number of ARs (Aggregation Routers) by dividing the traffic by a new parameter, the aggregation router capacity. Then the ARs are connected redundantly to all of the core routers in the PoP.

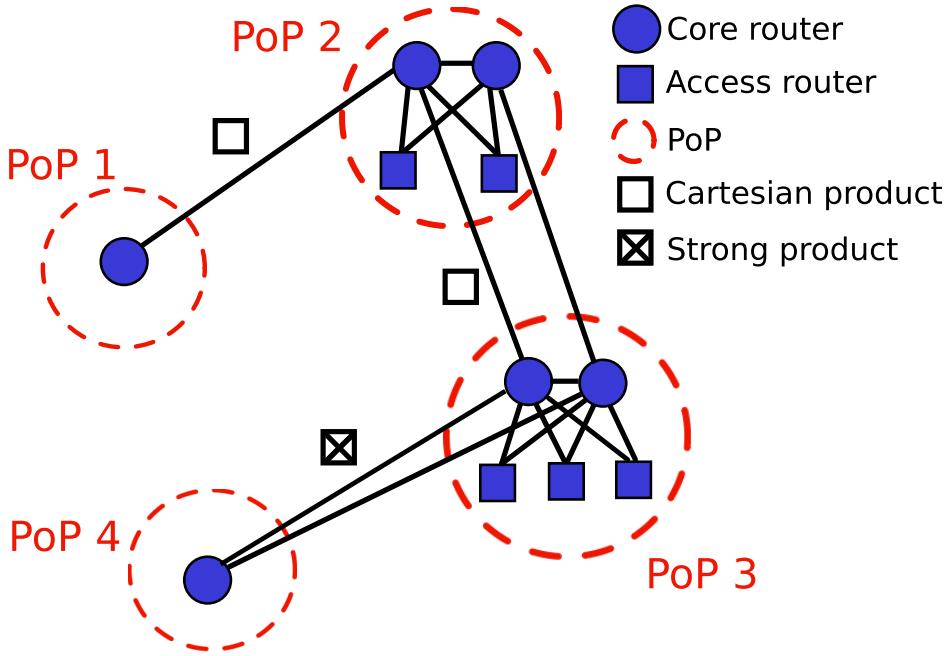


Figure 3.15: Each PoP shows the routers inside it and the graph product templates used to connect core routers of adjacent PoPs. PoPs with higher traffic require more access routers, and greater redundancy to nearby PoPs.

The process is shown in Figure 3.15, which shows four PoPs in red. The templates and products shown are applied to generate the core routers and the way ARs are attached to the core routers.

Note, although the model used to generate populations is inconsequential at the PoP-level, it makes a difference here. A power-law population model will result in many small and a few larger PoPs, potentially making a dramatic difference to the number of ARs, and this in turn affects the router-level node-degree distribution.

Graph products are a simple yet flexible method of synthesizing router-level topologies [57]. Here we gave a simple set of rules that embodied the choice of PoP templates and the requirement for redundancy with the addition of only two parameters. However, the technique is flexible enough to allow the formulation of complex engineering requirements. We have not yet explored all the possibilities that could be used to construct realistic router-level topologies, but leave this for future work.

3.9 Conclusion

This chapter presents COLD, an algorithm for generating synthetic data-network topologies motivated by real-world design approaches. It is as simple as possible, yet tuning the input parameters allows a wide variety of topologies to be produced mirroring those

of real-world networks.

One of the great benefits of this approach is that it allows for intuitive and sensible scaling. If small networks can be generated, so can larger networks, including networks with more nodes, spanning a larger area, carrying more traffic or some combination of these.

COLD is a conceptually simple model for synthesizing networks. It relies on a complex algorithm (optimization and graph products); but we provide an open implementation written in Matlab [68].

Now that we have a method for synthesizing a wide variety of realistic network topologies we can consider the reason we needed such a method: to allow thorough testing of our network tomography algorithms. The next chapter is dedicated to defining the network tomography problems we wish to solve, and proposing methods to solve them.

Chapter 4

Network link tomography and compressive sensing

4.1 Introduction

Accurate and timely performance data are of vital importance for network administration. However, modern networks are so large and transmit such enormous quantities of data that taking and processing all the desirable performance measurements can be wildly impractical. Aside from matters of scale there may be other difficulties, such as incomplete or unreliable measurement infrastructure. Consequently, it is necessary to make the most of the measurements that *are* available. *Network tomography* does just that, by inferring underlying performance statistics from the available measurements.

This chapter considers the problem of *link tomography*. That is, inference of link parameters from a series of end-to-end probes through a network – specifically, estimating average link loss rates. There are now many techniques for attacking this problem. Initially these techniques focused on tree-shaped network topologies, since these are the topologies induced by a single source probing a network [14]. However, the problem considered on a general network is more difficult because typical problems in that setting are highly underconstrained [9], and so the measurements admit many solutions (often infinitely many). Some method is needed to select the correct solution from this possible set, and in this chapter we test the idea of applying the (apparently ideally suited) field of compressive sensing to this link tomography problem.

Compressive sensing exploits the fact that many large data-sets are comprised of only a few significant elements. In practice, this means that either the data itself (represented perhaps by a vector of numbers), or some simple transform of the data, is *sparse* in the sense that only a few of the values are non-zero. In fact, the usual assumption is

that only a few values are significant, and the rest can be approximated as zero without important loss of information. In loss inference, sparsity can be applied because in well-run wired networks the loss rates should be small on most links. Many of the same ideas have been exploited for years in image and audio compression, but here we use them for estimation.

Compressive sensing is a rapidly growing area of research, but a key assumption in much of the area is that the experimenter controls the *measurement matrix*. In our context the measurement matrix is called a *routing matrix* and it is *not* chosen to suit the inference problem; its choice is mandated by the design of the network. What's more, we show here that routing matrices don't satisfy key properties that would allow us to directly apply the theory of compressive sensing.

For instance, routing matrices often have small groups of linearly dependent columns, and this means that we cannot even guarantee that a sparse solution to the measurement equations will be unique. The central question of this chapter is "Can we still use the concepts and methods of compressive sensing, despite the deficiencies of the routing matrices as measurement matrices?"

We show here that we can apply compressive sensing, with a reasonable degree of accuracy. It turns out that we can often find a unique sparsest solution. However, the existence of a unique solution does not just depend on conditions on the measurement matrix, but also on the measurements themselves, and this type of condition does not appear in the standard compressive sensing literature.

More importantly, the structure of typical routing matrices can be exploited. We develop here a new algorithm — Coherent Tomographic Deduction (CTD) — for solving the tomography problem and show that it is orders of magnitude faster than a standard compressive sensing technique, ℓ_1-norm minimisation [70], with the same accuracy.

Aside from being much faster, our algorithm has one other very significant advantage. It knows where it is definitely right, and where it could be wrong. The problem is decomposed in such a way that we can obtain exact results for some part of the network, and know, up to a single degree of freedom, the loss rates on other subsets of links in the network. This allows us to not just present estimates, but also to know which results are correct, and which may contain errors.

4.2 Background

We have presented a detailed background in Chapter 2. Here we just present a short overview. Network tomography concerns inference of network properties from indirect

measurements. Often, such measurements are the only ones available either due to expense, data volume, or impracticality of directly measuring the phenomena of interest. Here we are concerned with performance measurements, where ideally, a network operator would like to know performance statistics for every link in a network. However, there can be many thousands of links in a large network, and so such measurements are often impractical. Instead, one makes do with end-to-end measurements of performance along a set of paths through the network, and from these infers the performance on the links.

Network tomography problems are often hard because they are underconstrained [9,21], and that is the case here. We don't have enough information to unambiguously infer the link performance statistics from the path measurements. Solutions to these problems therefore require side-information of some form. In our case we exploit sparsity.

Network tomography is a well developed field at this point of time (see for example [7, 10–12]). Much performance tomography has concentrated on trees. In that setting, it is possible to develop fast, recursive algorithms [11,13], and to employ side information such as sparsity relatively easily. For instance, Duffield [14] uses sparsity to resolve the location of "bad" links. In a tree, he shows that the sparsest solution will have "bad" links as high as possible in the tree. Arya and Veitch [15] use the information given by link loss rates to show that the sparsest solution on a tree topology is one where the higher links take as much loss as possible. Both these algorithms consider links in terms of their (hop-count) distance from the source, and assign as much loss as possible to each link in the order. That is, find the paths running through each link, pick the path with the lowest loss rate and assign all remaining loss to that link.

However, many networks are not trees. Extending those tree approaches to topologies with more than one source is problematic since there is no longer a natural ordering on the links, and applying either previous method is likely to produce a solution that is either inconsistent with the measurements or not the sparsest possible solution. Network tomography on these networks still needs to employ additional information to avoid the problem remaining undetermined. One approach is to use multicast probes [17, 18]. In environments without multicast enabled, this approach can be emulated using a string of unicast probes sent back-to-back [11, 19, 20]. Another approach is to use several periods of probing to determine the covariances of the path loss rates, then use this to determine the variances of the link loss rates. Links with very low variance in loss rate are then assumed to have few losses, and this can greatly reduce the number of unknown links, making the tomography equations fully determined [9, 21].

In this chapter we attack the problem of network-performance tomography on a general network. We also exploit sparsity, though we do it without reducing the problem to a binary (good/bad link) problem. Padmanabhan *et al.* [70] also used the idea of finding a tomographic solution with the minimal number of lossy links. However our algorithm

exploits the properties of routing matrices to gain two large advantages: it is much faster without losing any accuracy; and it supplies additional information - it labels which links are guaranteed to be correct, and which links are merely best estimates.

4.2.1 Notation and assumptions

We consider a network described by the graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of (directed) edges, with the number of links given by $|\mathcal{E}| = n$. We have measurements from m paths, labelled q_j, $j = 1, \ldots, m$.

We consider here the problem of inferring the transmission probabilities on links in a network from path measurements, and define (as in Chapter 2)

$$\begin{aligned} t_i &= \text{link transmission probability for } i = 1, \ldots, n, \\ p_j &= \text{path transmission probability for } j = 1, \ldots, m. \end{aligned}$$

When losses on different links are independent the two are simply related by

$$p_j = \prod_{e_i \in q_j} t_i, \quad \text{for all } j = 1, \ldots, m.$$

We take the log of the above equations in order to get a linear relationship between the variates, and write them in matrix form as

$$\boldsymbol{\rho} = A\boldsymbol{\tau} \tag{4.2.1}$$

where

$$\begin{aligned} \tau_i &= -\log t_i, \quad i = 1, \ldots, n, \\ \rho_j &= -\log p_j, \quad j = 1, \ldots, m, \end{aligned}$$

are the *loss coefficients* and A is the $m \times n$ routing matrix defined by

$$A_{j,i} = \begin{cases} 1 & \text{if link } i \text{ is on path } j; \\ 0 & \text{otherwise.} \end{cases}$$

Note that we work with the loss coefficients, $\tau_i, \rho_j \in [0, \infty]$. $\tau_i = 0$ corresponds to 0% loss probability on link i, increasing with increasing loss up to $\tau_i = \infty$ corresponding to 100% loss probability; similarly for ρ_j with respect to path performance.

We assume that the routing matrix A is known. Our approach is agnostic as to whether multicast or unicast measurements are used. We do not exploit correlations in loss in specific probes, but rather consider the statistical averages over a set of measurements,

and so avoid the need to have probes that are strongly correlated in time. We also avoid the need to have multiple measurement periods, to correlate losses on paths, as in [22]. We assume for now that (4.2.1) is consistent. This corresponds to assuming there is no noise in the measurement system, an assumption we will weaken in Chapter 6.

The goal here is to find the $n \times 1$ vector $\boldsymbol{\tau}$ from the m path measurements $\boldsymbol{\rho}$. The problem would be simple if A were full-rank, as the problem is then just one of solving linear equations. However, A is rarely full-rank. Even when $m > n$, it is common that redundancy in the measurements leads to an underconstrained set of equations with infinitely many solutions. To address this we exploit sparsity.

4.3 Sparsity

In general, sparsity is associated with large datasets in which only a few elements are non-zero, either in the original dataset, or in a transformed version of that data. Sparse datasets occur in several settings, including image analysis [71], group testing [72] and DNA microarrays [73]. In the network tomography setting, sparsity has already been exploited [14]. The assumption is that in a well run (wired) network, most links will be "good" most of the time, where "good" refers to the link having a negligible loss rate. In that setting, the goal of network tomography is to identify the set of poorly performing links, and perhaps prioritize action on them.

We can capture the notion of sparsity with the following definition:

Definition 2. *We say that a vector* $\boldsymbol{\tau}$ *is* S-sparse *iff* $||\boldsymbol{\tau}||_0 \leq S$ *where* $||\cdot||_0$ *denotes the* ℓ^0 *norm,* i.e., $||\mathbf{x}||_0$ *counts the number of non-zero elements of* $\mathbf{x}$.

Aside from the fact that real sets of data are sparse, why is sparsity useful? Since we are looking for ways to reduce infinitely many solutions down to the one physical solution, we might naturally ask "Does the search for sparsest solutions lead to a unique solution?" There is now a large literature — often going under the heading of *compressive sensing* — on exploitation of sparsity in inference problems. The following is a well known result justifying the idea that introducing sparsity can induce a unique solution.

Theorem 2 (Uniqueness [23])**.** *Let* Φ *be a matrix with every set of* $2S$ *columns linearly independent. Then for any measurement vector* $\boldsymbol{\rho}$*, the system* $\boldsymbol{\rho} = \Phi\boldsymbol{\tau}$ *has* at most one S*-sparse solution for* $\boldsymbol{\tau}$.

That is to say, if we know that our routing matrix obeys the above property, and the solutions of interest are sufficiently sparse (S-sparse), then we know there will be a unique solution corresponding to our measurements.

If we associate the measurement matrix Φ with our routing matrix A, and note that our desired vector $\boldsymbol{\tau}$ is sparse, then the above theorem appears directly applicable. We might try finding $\boldsymbol{\tau}$ by simply solving the constrained optimisation problem:

$$\min_{\tau} ||\boldsymbol{\tau}||_0 \text{ subject to } \boldsymbol{\rho} = \Phi\boldsymbol{\tau}. \tag{4.3.1}$$

In other words "find the sparsest solution consistent with the measurements". In general, a direct solution to this problem is a combinatorial optimisation problem, and the worst case run-time grows very quickly with respect to $||\boldsymbol{\tau}||_0$. However, the compressive sensing literature now has a wealth of algorithms for solving these problems efficiently [24–27].

To guarantee the performance of these compressive sensing algorithms the measurement matrix must satisfy appropriate conditions. One criterion commonly used is the *Restricted Isometry Property* (RIP, [28]). Routing matrices have only 0 or 1 entries, so the corresponding property is *RIP-1* [24]. To generate matrices that satisfy *RIP-1* with high probability, it is simply necessary to generate a 0-1 matrix uniformly at random such that each column has the same number of ones. Such 0-1 matrices require more measurements asymptotically than ideal non-integer matrices [29] but are still satisfactory for our application.

Xu *et al.* [74] and Mahyar *et al.* [75] present methods for performing network tomography using compressive sensing. Xu *et al.* show that if a graph is *sufficiently connected* and routes are chosen by sufficiently long random walks on that graph, then $O(k\log(n))$ measurement routes give a routing matrix that allows for recovery of any k-sparse link vector. They use ℓ_1-minimization to recover such a link vector. Mahyar *et al.* [75] improve upon this work by choosing more efficient random walks through the graph. The first problem with applying these results in practice is that the theorems are asymptotic; the guarantees they give in finite cases are overly conservative. Both papers address this by providing simulation results that show good performance on reasonably-sized graphs. The second problem is that a real network may not be sufficiently connected, or the routes may not be sufficiently long. The most important problem however is that the experimenter typically has almost no control over routing matrices; in practice routes will *not* be determined by random walks on the network. For instance, if using shortest-path routing, then any routes that pass through node A and node B will all take the same route between A and B. In practice, the quality of the routing matrix as a measurement matrix is far worse than those in [74] and [75].

Firooz and Roy [76] also use compressive sensing to produce theoretical guarantees for performing network tomography. They consider more realistic routing matrices than [74] and [75], using shortest paths between nodes around the edge of a network. They then show that ℓ_1-minimization will produce the correct topology under the assumptions that the bipartite graph representation of the routing matrix is a union of appropriate

expander graphs. Unfortunately, this is not a property we can guarantee in practice: we find it rarely holds for those routing matrices used in testing in the later chapters of this thesis.

Typically routing matrices are poor measurement matrices. The first problem is that when using routing matrices as measurement matrices there is no guarantee that a unique solution even exists, so it may be impossible to find one. The second problem is that even if such a solution exists, routing matrices typically don't satisfy RIP-1, and hence the success of compressive sensing algorithms is not guaranteed. In the two following sections we explain why routing matrices have these two problems.

4.4 Existence of unique sparsest solutions

The first and biggest issue to confront is establishing whether or not there is a unique sparsest solution to the system of measurement equations. The answer is, in general, no. Theorem 2 cannot be applied because routing matrices frequently contain small subsets of columns that are linearly dependent (see the examples below). Despite this, a unique sparsest solution to the measurement equations can still exist, but there is no guarantee of such.

One example of when a sparsest solution is not necessarily unique is when we have *indistinguishable links*. These are two (or more) links that occur on exactly the same measurement paths. Consequently, any measurement that includes either link includes the other link. If loss occurs on one of these two links then there is no way to determine which link it is using just the path measurements. This is a common case in the literature, and it is a standard step at the start of any problem to aggregate each group of indistinguishable links into one *logical link*. Indistinguishable links are easy to spot in the routing matrix; they correspond to identical columns[1].

Unfortunately, indistinguishable links are not the only problem in finding a sparsest solution. We can start by working with Theorem 2. We rephrase this as: *Let D be the smallest subset of columns that is linearly dependent, and let $C_A = |D|$, the number of columns in that subset. If we can find a vector $\boldsymbol{\tau}$ with fewer than $C_A/2$ nonzero entries that satisfies $\boldsymbol{\rho} = A\boldsymbol{\tau}$, then that vector is the unique sparsest solution.* If we know an upper limit on the number of nonzero entries in $\boldsymbol{\tau}$ (*i.e.,* the max number of lossy links), then Theorem 2 gives a condition on the routing matrix sufficient to guarantee a unique sparsest solution.

Figure 4.1 shows a very simple topology as an example. The routing matrix for this

[1]In the routing matrix each column corresponds to a particular link and each row to a path. If two columns are identical, the two corresponding links appear on exactly the same paths.

$\boldsymbol{\rho}$	sparsest $\boldsymbol{\tau}$	$\|\|\boldsymbol{\tau}\|\|_0$
$[3,3]^T$	$[0,0,3]^T$	1
$[2,3]^T$	$[0,1,2]^T, [2,3,0]^T$	2

Table 4.1: Alternative scenarios for Figure 4.1

topology is

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

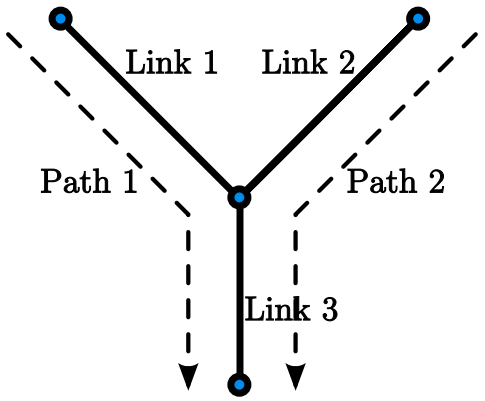


Figure 4.1: Simple Y-shaped topology example.

Any two columns in the routing matrix are linearly independent (but not all three) so $C = 3$. Consequently, if the solution $\boldsymbol{\tau}$ has $||\boldsymbol{\tau}||_0 < 1.5$ then it is the unique sparsest solution. For an example, see the first row of Table 4.1. In this case $\boldsymbol{\rho} = [3,3]^T$ and $\boldsymbol{\tau} = [0,0,3]^T$. $||\boldsymbol{\tau}||_0 = 1$ so there can be no other equally sparse solution (or sparser solution).

When there are no sufficiently sparse solutions (*i.e.,* ones with $||\boldsymbol{\tau}||_0 < 1.5$) then uniqueness is not guaranteed. See the second row of Table 4.1 for example; where $\boldsymbol{\rho} = [2,3]^T$. The same network now has two maximally sparse solutions consistent with these measurements, $[0,1,2]^T$ and $[2,3,0]^T$.

So there are several difficulties with the condition in Theorem 2. First, in its current form this condition is difficult to check efficiently as C_A grows. Second, we don't control the measurement matrix. Third, this parameter C_A (the least number of linearly dependent columns) is generated from a *worst case* over all subsets of columns. Ideally we want C to be as large as possible, but any "bad" subset of columns can ruin this for us, and obscure the benefit we might be getting in the remainder of the matrix. This means that while the condition based on C_A is sufficient to guarantee a unique sparsest solution, it is not necessary. As a specific example of the condition in Theorem 2 being

i	$\boldsymbol{\rho}^{(i)}$	sparsest $\boldsymbol{\tau}$	$\|\|\boldsymbol{\tau}\|\|_0$
1	$[2,2,2,0,0]^T$	$[2,2,0,0,0,0]^T, [0,0,2,2,0,0]^T$	2
2	$[1,0,0,2,1]^T$	$[0,0,0,0,1,1]^T$	2

Table 4.2: Alternative scenarios for example 2, with routing matrix A_2

too weak, consider the routing matrix

$$A_2 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{4.4.1}$$

The first four columns of A_2 are linearly dependent so the value of C_{A_2} is at most 4 (it is in fact 4). In a sense these four links ruin C for the rest of the matrix. Considering the measurements $\boldsymbol{\rho}^{(1)}$ in Table 4.2, there are two sparsest solutions $\boldsymbol{\tau} = [2,2,0,0,0,0,0]^T$ and $[0,0,2,2,0,0]^T$. This is caused by the four linearly dependent columns of A_2.

Now consider the measurements $\boldsymbol{\rho}^{(2)}$. The solution $\tau = [0,0,0,0,0,1,1]^T$ is the unique sparsest solution in this case, despite having greater than or equal to $C_A/2 = 2$ nonzero elements. Part of the reason this solution is unique despite not satisfying the sufficient condition from Theorem 2 is that links 1 to 4 (columns 1 to 4) are not included in the solution (the first four elements of $\boldsymbol{\tau}$ are both zero). This linearly dependent subset of links constrains the value of C for the whole matrix, but isn't relevant to the uniqueness of the second solution.

There are two things to be learnt from this example. First, the condition given in Theorem 2 is not tight in general; it is not a necessary condition for a unique solution to exist. Whether or not a sparsest solution exists can depend on the actual measurements made, and not just the routing matrix A

Second, it is possible to find a unique solution for some of the links, without finding one for all of the links, *e.g.*, links 5 and 6 in the first row of Table 4.2. We say that a subset of the links $I \subset \mathcal{E}$ has a unique sparsest solution if all of the sparsest solutions for $\boldsymbol{\tau}$ must have the same loss coefficient on I.

So we can see that it is not easy to characterize the existence of unique sparsest solutions. However, there is another problem to deal with: Do routing matrices have the properties required by compressive sensing algorithms, even when a unique solution exists?

4.5 Routing matrices are poor measurement matrices

To answer the question posed above, we need to ascertain whether routing matrices are good measurement matrices. What makes a good measurement matrix? One widely used condition is RIP, or in the case of 0-1 matrices, RIP-1 [24]. Berinde *et al.*[24] prove that a sensing matrix satisfies RIP-1 with "good" parameters iff its associated graph is a "good" expander. In the routing matrix case, this is equivalent to each small subset of paths covering a relatively large number of links. Let P be a set of paths, let $\mathcal{L}(P)$ be the set of all the links on those paths. Then for a routing matrix to be a "good" expander, it must satisfy $|\mathcal{L}(P)| \geq k|P|$ for all P such that $|P| \leq M$ for as high values of k and M as possible. Ideally k should not be much less than the minimum number of links on a path (0-1 matrices designed for compressive sensing typically have the same number of ones in each column, *i.e.,* the same number of links on each path).

To see that routing matrices are not good expanders, consider the *forwarding principle*: if the path from node A to node C in a network goes through node B, then it contains the path from A to B and the path from B to C. That is, any subpath of a path is also a path. Now consider a path with n links. Such a path will have $\binom{n+1}{2}$ subpaths. All of these $\binom{n+1}{2}$ subpaths will be paths in their own right, and the set containing all these paths will only measure n links. This gives very poor expansion properties, and a k of much less than 1.

So routing matrices typically have poor expansion properties and don't satisfy RIP-1 or similar properties. Nevertheless, the conditions above are only sufficient conditions. In the remainder of this thesis we show that we can still use sparsity to accurately estimate solutions to link tomography problems, despite the difficulties described above.

4.6 Algorithm Components

There are a number of methods we use to attack this problem. Some steps will solve for some variables; other steps will decompose the problem, producing one or more new sub-problems $\boldsymbol{\rho}^{\text{new}} = A^{\text{new}}\boldsymbol{\tau}^{\text{new}}$ from a set of equations $\boldsymbol{\rho} = A\boldsymbol{\tau}$, and these new problems can in turn have reductions applied to them. We will first describe the components, and then how they combine to form an algorithm.

Indistinguishable links step: It will occasionally be the case that there are links that are impossible to distinguish, given the set of measurements provided, *e.g.*, two or more links that occur on exactly the same paths. This situation is represented in the routing matrix as two identical columns. We cannot distinguish the variables but

we can still attempt to solve for them as a whole. Hence, our first step is to aggregate each group of indistinguishable links into a *logical link*. This step is common in the link tomography literature [14]. Note that we need to apply this reduction only once, since once indistinguishable links are aggregated, none of the following reductions can cause them to reoccur in the reduced problems.

An efficient implementation of this step is to sort the n columns as length-m binary strings, then compare adjacent columns. The run time for this is $\mathcal{O}(mn \log n)$, assuming comparisons are made in time $\mathcal{O}(m)$. This is close to optimal, since the number of elements of A is mn, and we must inspect all of them in the worst case.

***Single-link paths* step:** If a path q_j traverses only one link then that link must be responsible for all the loss on that path. This corresponds to a row of A with a single element 1 and the rest of the elements 0. That row represents the equation

$$0\tau_1 + 0\tau_2 + ... + 0\tau_{i-1} + 1\tau_i + 0\tau_{i+1} + ... + 0\tau_n = \rho_j,$$

$$\text{that is, } \tau_i = \rho_j.$$

Thus we can immediately solve for that link, and remove both it and the path containing it from our system of equations. This step has runtime of $\mathcal{O}(mn)$.

***Zero-loss paths* step:** The next step is to consider paths with no loss. Say path q_j has no loss, *i.e.,* $\rho_j = 0$. Then:

$$\sum_{i:A_{j,i}=1} \tau_i = 0, \quad \tau_i \geq 0.$$

Hence

$$\tau_i = 0 \quad \text{for all } i : A_{j,i} = 1.$$

This *zero-loss paths* step can remove more equations and variables from the system under consideration, further simplifying it. It runs in time $\mathcal{O}(mn)$.

Iteration: Both the *single-link paths* step and the *zero-loss paths* step, in reducing the problem to a simpler one, can also have the side-effect of recreating the conditions required for the other step. For instance, the *single-link paths* step can reduce the elements of $\boldsymbol{\rho}$ so that it has new zero entries. Also, the *zero-loss paths* step can remove variables, leaving paths with only one unsolved link. Consequently, we iterate over these two steps until either we have a full set of solutions, or neither step can go any further.

For the iteration to continue, each step must remove at least one equation. Since we start with m measurement equations, this process of iteration cannot continue for more than $m + 1$ steps, and it usually terminates much sooner.

Before introducing the next step, we define bipartite graphs and present two related lemmas.

Definition 3 (Bipartite graph)**.** *A graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$ is bipartite if $\exists$ two sets of nodes U, V such that $U \cup V = \mathcal{N}$, $U \cap V = \emptyset$ and every edge $(x, y) \in \mathcal{E}$ has $x \in U, y \in V$ or $x \in V, y \in U$.*

The following two lemmas give another characterisation of connected bipartite graphs. The proofs for these are well known, see for example [77].

Lemma 3. *Let i be a node in a connected bipartite graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$ with the sets U and V as above. Without loss of generality let $i \in U$. Then U is the set of nodes that can be reached by even length paths from i, and V is the set of nodes that can be reached by odd length paths from i.*

Lemma 4. *Let i be a node in a connected non-bipartite undirected graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$. Then $\exists j \in \mathcal{N}$ such that there is both an odd length path and an even length path from i to j.*

Splitting step: At any point in the process we can apply a *divide and conquer* approach; that is, split the problem into independent subproblems and solve those subproblems, then combine the solutions. Since solution times for (4.3.1) rise superlinearly in n, then the sum of the times taken to solve the subproblems is frequently much less than that taken to solve the overall problem.

If the overall system of equations can be partitioned into sets of equations with no variables in common, then the problem can be divided into subproblems according to these partitions. Each set of equations involves a specific set of variables[2], so this is

[2]We say an equation *involves* a variable if that variable has a non-zero coefficient in the equation.

equivalent to partitioning the set of variables into disjoint subsets, where any equation involves variables from exactly one partition. A fast method to find these partitions is to create a bipartite graph $\mathcal{G}_1(\mathcal{N}_1, \mathcal{E}_1)$ from the routing matrix. We let the nodes in U represent paths and the nodes in V represent links. If $j \in U, i \in V$ then $(i, j) \in \mathcal{E}$ iff $A_{j,i} = 1$, that is, if link i is on path j. The independent subproblems just correspond to the connected components of this graph. Connected components of a graph can be found efficiently in time $O(|\mathcal{N}_1| + |\mathcal{E}_1|)$; where $|\mathcal{N}_1|$ in this case is the number of paths plus number of links, $m + n$; and $|\mathcal{E}_1|$ is the sum of the number of links in all the measurement paths. We use this approach to split the problem into smaller subproblems.

Two-link paths step: What is there to do once the zero-loss paths and the single-link paths steps have run their course? As we have dealt with all the paths with only one link on them, it makes some sense to consider paths with up to 2 links. Note that here we are temporarily disregarding some information, but it is quite possible that we can nonetheless make some progress. Since the original equations are consistent, if we can find a solution for this reduced system of equations it must be part of any solution of the overall system. So we temporarily ignore all paths with more than 2 links. All remaining paths have some loss and exactly 2 links. These paths are partitioned using splitting, as above. Later we will use the fact that the resulting equations cannot be further split. Consider one of these subproblems,

$$\boldsymbol{\rho}^* = A^* \boldsymbol{\tau}^*. \tag{4.6.1}$$

Now one of two things is true: either we can find a unique solution for the entirety of this subproblem, or we can find a set of solutions with only one degree of freedom. We will prove this and give the conditions under which a unique solution is possible.

Define another graph by associating a weighted undirected graph $\mathcal{G}_2(\mathcal{N}_2, \mathcal{E}_2)$ with Equation (4.6.1). Each node in $\mathcal{N}_2$ corresponds to a variable in our set of equations, and each edge in $\mathcal{E}_2$ corresponds to an equation. Each edge is incident with two nodes, these correspond to the two variables in that edge's equation. For instance, if the j^{th} equation is given by

$$\tau_{i_1} + \tau_{i_2} = \rho_j$$

then there is an edge between nodes i_1 and i_2 with weight ρ_j.

Theorem 5. *Equations (4.6.1) are uniquely solvable if $\mathcal{G}_2$ is not bipartite. Otherwise, they are solvable with one degree of freedom.*

Proof. $\mathcal{G}_2$ represents a set of equations with one variable for each node, and one edge for each equation. Let $\{\tau_i\}_{i \in N}$ be the set of variables. We start by letting $\tau_{i^*} = t$ for

some arbitrary $i^* \in \mathcal{N}_2$. Consider a node adjacent to node i^* in $\mathcal{G}_2$, node j say. Then since it is adjacent to i^* it is connected by an equation

$$\begin{aligned} &\tau_{i^*} + \tau_j = \rho_r \quad \text{for some } q_r \in \mathcal{E}_2, \\ &\text{which implies that } \tau_j = \rho_r - \tau_{i^*} = \rho_r - t. \end{aligned}$$

This gives us an expression in one variable (t) for each of the nodes adjacent to i^*. We can extend this to nodes adjacent to each of these nodes in turn. For example, if we have

$$\tau_{j_1} = \rho_{r_1} - t,$$

and node j_2 is connected to node j_1 via edge r_2 then

$$\begin{aligned} \tau_{j_1} + \tau_{j_2} &= \rho_{r_2} \\ \text{so } \tau_{j_2} &= \rho_{r_2} - \tau_{j_1} \\ &= \rho_{r_2} - (\rho_{r_1} - t) \\ &= (\rho_{r_2} - \rho_{r_1}) + t. \end{aligned}$$

This is similar to before, but with a positive coefficient for t. In turn, we can find an expression for any node connected to this second set of nodes, where t has a negative coefficient. Each time we navigate a link the coefficient for t changes sign.

Let q represent a path in our new graph $\mathcal{G}_2$. The number of edges in q is $|q|$. Let $Q(k)$ be the k^{th} node on path q, for all $k = 0, \ldots, |q|$. So the k^{th} edge of q is $(Q(k-1), Q(k))$. Consider an arbitrary node $i_1 \in \mathcal{N}_2, i_1 \neq i^*$. Since the graph is connected (because of the splitting step above), there is at least one path from i^* to i_1. Call one such path q. Using this path q and repeating the process above, we can find an expression for τ_{i_1}:

$$\tau_{i_1} = (-1)^{|q|} t + \sum_{j=0}^{|q|} (-1)^{j+|q|} \rho_{(Q(j-1),Q(j))}, \tag{4.6.2}$$

where $\rho_{(Q(j-1),Q(j))}$ is just the constant associated with the j^{th} edge along q. Since $\sum_{j=0}^{|q|} (-1)^{j+|q|} \rho_{(Q(j-1),Q(j))}$ is just a constant, each expression just looks like

$$\tau_{i_1} = (-1)^{|q|} t + c$$

for some c, where $|q|$ is the length of a path from i^* to i_1. If for any node $i_0 \in N$ there exists an odd length path and an even length path, then at that node there will be at

least two expressions for τ_{i_0} with different coefficients of t, say

$$\begin{aligned} \tau_{i_0} &= c_1 - t \\ &= c_2 + t. \end{aligned}$$

Then we can solve for t, giving $t = \frac{c_1 - c_2}{2}$. This in turn allows us to solve for each other variable τ_i by substituting t into the appropriate expression. Thus, if there exists a node that can be reached by both an odd length path and an even length path from i^*, then we can solve the equations uniquely for all τ_i, $i \in N$. Using Lemma 3 and Lemma 4 we can see that such a node exists iff the graph is not bipartite.

If no such node exists, that is, there is no node that is both an odd and even length from i^*, then G is bipartite and we cannot solve the equations uniquely because any expressions we find will be $c_i + t$ for nodes an even distance away, and $c_i - t$ for nodes an odd distance away. These expressions are consistent so we cannot use them to solve for t. □

This proof suggests a method for finding these solutions. Ignore all but the length 2 paths, then split into subproblems, and either we can use the above theorem to solve for the τ_i or we can write a set of inequalities based on the degree of freedom t for each subproblem.

At the end of this *two-link paths* step we incorporate any solutions found into the top level solution. For each subproblem where the link values are determined we can eliminate those links and paths from the overall problem. For each subproblem where the links are determined up to a single degree of freedom we eliminate all the paths and all but one link from the overall problem. This will potentially lead to new single-link or zero-loss paths, and so we iterate through these steps until no more changes occur. We can then introduce sparsity.

Introduce sparsity: While ℓ_1-minimization is far more efficient than finding the sparsest solution directly by ℓ^0-minimization, it is still the most computationally intensive step. Hence we wait as late as possible to do it, when we have reduced the size of the problem.

Now that we have reduced the system of equations to a simpler form, we can attempt to find the sparsest consistent solution. Once again we split the system (only this time we use all the paths, not just those of length 2). If we can separate the problem into multiple independent subproblems, then we can find the sparsest solution to each of these subproblems separately; the overall answer will just be the combination of the answers to the subproblems. At this point we have a choice of methods to try to find the sparsest solution: we can attempt to find it in the simplest way, by doing combinatorial optimisation; or we can try applying compressive sensing techniques such as ℓ_1-norm

minimisation [23, 25, 78] or Orthogonal Matching Pursuit [27, 79]. In this chapter we choose to use ℓ_1-norm minimisation.

ℓ_1-minimisation replaces the problem of finding the sparsest solution to our set of equations,

$$\min_{\tau} ||\boldsymbol{\tau}||_0 \text{ subject to } \boldsymbol{\rho} = A\boldsymbol{\tau}, \tag{4.6.3}$$

with the computationally simpler problem

$$\min_{\tau} ||\boldsymbol{\tau}||_1 \text{ subject to } \boldsymbol{\rho} = A\boldsymbol{\tau}. \tag{4.6.4}$$

Problem (4.6.4) can be solved using linear programming, and a wealth of literature [23, 24, 78, 80] indicates that its solution will be the same as the solution to Problem (4.6.3) under certain conditions on A and with sufficiently sparse $\boldsymbol{\tau}$. While we have pointed out that these conditions do not hold in general for routing matrices A, we can still hope to use the method to approximate sparsest solutions. It is common in signal processing to use algorithms in situations where they can't be rigorously proven to work; for instance Kalman filtering without the guarantee of Gaussian noise in the measurements. In Chapter 5 we test whether ℓ_1-minimisation will provide a good estimate of the remaining undetermined variables. Applying the *single-link paths*, *zero-loss paths* and *two-link paths* steps before running ℓ_1-minimisation has two benefits: speeding up the process and providing certainty about the values of some of the variables.

4.6.1 Naïve Algorithm

The previous subsection provides the detail of an algorithm. In Algorithm 1 we show the overall structure of our initial (naïve) approach.

Algorithm 1
Stage 1

- Aggregate indistinguishable links.
- *While* the solution is changing
  1. *While* the solution is changing
     (a) **Single-link paths** Solve for each link that is on a path by itself, then substitute this value into the other equations.
     (b) **Zero-loss paths** Solve for each link that is on a path with no loss, and substitute this value into the other equations.

     end While
  2. **Two link paths:**
     – Take the system corresponding only to those paths with two links.
     – Split this system into independent subproblems.
     – Solve for each subproblem, possibly leaving 1 degree of freedom.
     – Combine the solutions.

  end While

Stage 2

- **Splitting:** Split the system into independent subproblems.
- **Sparsity:** Apply ℓ_1-norm minimisation to each of the remaining unsolved subproblems.

The above approach is reasonable, but closer examination of Step 1 shows we can do better. We see in the next section that the *single-link paths* and *zero-loss paths* steps can be built into a more general algorithm for Step 1.

4.7 Coherent Tomographic Deduction (CTD)

We can improve the previous approach and we call the resulting algorithm Coherent Tomographic Deduction (CTD).

4.7.1 Stage 1

If we examine each of the steps in Algorithm 1 individually, we can see that only one of the steps uses the non-negativity of the variables τ_i. That step is the *zero-loss paths* step. This suggests the question: can we get more from the non-negativity condition on $\boldsymbol{\tau}$?

For example, consider the system $A\boldsymbol{\tau} = \boldsymbol{\rho}$ with

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \text{ and } \boldsymbol{\rho} = \begin{bmatrix} 2 \\ 5 \\ 3 \end{bmatrix}.$$

Start by considering the first row, corresponding to $\tau_1 + \tau_2 = 2$. Now since $\tau_1 \geq 0$, we know that $\tau_2 \leq 2$. Also, $\tau_3 \leq 3$ because of the third row. However, $\tau_2 + \tau_3 = 5$ (second row) and so $\tau_2 = 2$ and $\tau_3 = 3$. Algorithm 1 would only discover this in Stage 2, but we can change the algorithm to make such inferences directly and efficiently.

Iterative Bounding: For each variable we keep track of a lower bound and an upper bound, and use these to improve the bounds on the other variables. For instance, in the previous example start with the bounds

$$\tau_1, \tau_2, \tau_3, \tau_4, \tau_5 \geq 0.$$

Then in our first step we derive

$$\tau_1, \tau_2 \leq 2, \tau_2, \tau_3 \leq 5, \tau_3, \tau_4, \tau_5 \leq 3.$$

Some of these bounds are redundant, only the strongest bound need be kept in each case, so

$$\tau_1, \tau_2 \leq 2 \text{ and } \tau_3, \tau_4, \tau_5 \leq 3.$$

Then since $\tau_2 = 5 - \tau_3$ and $\tau_3 \leq 3$, we get $\tau_2 \geq 2$. From the first step $\tau_2 \leq 2$, so $\tau_2 = 2$. In general, we alternate between finding upper bounds for variables based on the previous lower bounds, and finding lower bounds for variables based on the previous upper bounds.

Let $\mathbf{l}^t$ be the length n vector of lower bounds attained after t steps. Similarly, let $\mathbf{u}^t$ be the upper bounds attained after t steps. Start with $\mathbf{l}^0 = \mathbf{0}$. Then define the rest of the sequence by

$$\mathbf{u}_j^{t+1} = \min_{i:A_{i,j}=1} \left(\rho_i - \sum_{k:A_{i,k}=1, i \neq k} \mathbf{l}_k^t \right), \quad t > 0, \tag{4.7.1}$$

$$\mathbf{l}_j^{t+1} = \max_{i:A_{i,j}=1} \left(\rho_i - \sum_{k:A_{i,k}=1, i \neq k} \mathbf{u}_k^{t+1} \right), \quad t > 0. \tag{4.7.2}$$

These two steps use the previous lower (upper) bounds and each equation to find the new upper (lower) bounds. Halt this process when $||\mathbf{l}_t - \mathbf{l}_{t-1}|| < \varepsilon$ for some sufficiently small $\varepsilon > 0$, that is, when the process has reached a stable state. Since $\mathbf{l}_t \leq \mathbf{u}_t$, and $\mathbf{u}_{t+1} \leq \mathbf{u}_t$, $\mathbf{l}_{t+1} \geq \mathbf{l}_t$ for all t (this can be proved by induction on t for u_t and l_t simultaneously) it must terminate in at most $||\rho_j||_1/\varepsilon$ steps. In practice it terminates *much* faster.

We can now replace Step 1 in Stage 1 of our algorithm with this iterative bounding approach, because it performs the *zero-loss paths* and *single-link paths* steps implicitly. We can see this as follows:

1. Zero-loss paths: Any path i that has no loss on it, *i.e.,* $\rho_i = 0$ will have $\mathbf{u}_j^1 = 0$, and $l_j^0 = 0$.
2. Single-link paths: If path i has just one link then $\{k : A_{i,k} = 1, i \neq k\} = \emptyset$. Consequently $\mathbf{l}_i^1 = \boldsymbol{\rho}_i - 0 = \boldsymbol{\rho}_i$. Also, $\mathbf{u}_i^1 = \boldsymbol{\rho}_i$, and $\mathbf{l}_i^1 = \mathbf{u}_i^1$

However iterative bounding does more than just apply these two steps. It also allows us to find the loss rates on some other links, as we can see from the previous example. Stage 1 of Algorithm 2 now consists of alternately applying the *Iterative Bounding* and *Two-link paths* steps.

4.7.2 Stage 2

While Stage 1 derives solutions for part of $\boldsymbol{\tau}$ with absolute confidence, there may be some components about which it is unsure. To estimate the remainder of the solution after running Stage 1, we use ℓ_1-minimisation, just as in Algorithm 1.

4.7.3 Coherent Tomographic Deduction

We replace Algorithm 1 with an improved algorithm: Coherent Tomographic Deduction (CTD).

Algorithm 2 - Coherent Tomographic Deduction
Stage 1

- Aggregate indistinguishable links.

While The solution is changing

- **Iterative Bounding:** As described in Equations (4.7.1) and (4.7.2).
- **Two-link paths:** As in Algorithm 1.

End while
Stage 2

- **Splitting:** Split the system into independent subproblems.
- **Sparsity:** Use ℓ_1-minimisation to estimate the remaining link loss values.

4.8 Computation Time Comparisons

CTD Stage 1 exploits features of routing matrices to try to increase the speed and certainty of ℓ_1-minimisation in finding link loss values. There is no guarantee that Iterative Bounding, Two-link paths and Splitting make CTD faster than direct methods. However, in practice, CTD Stage 1 greatly decreases the time spent by ℓ_1-minimisation in finding the estimate for $\boldsymbol{\tau}$. This can be seen in Figure 4.2, which compares the runtimes for 50 node networks on a logarithmic scale. CTD is much faster, particularly for a large numbers of measurement nodes, and its runtime also grows much more slowly than the raw ℓ_1-minimisation.

Not only is it much quicker, but CTD provides additional information: we know which of the links we are sure about, and which ones are merely an estimate, while raw ℓ_1-minimisation merely provides estimates for all links.

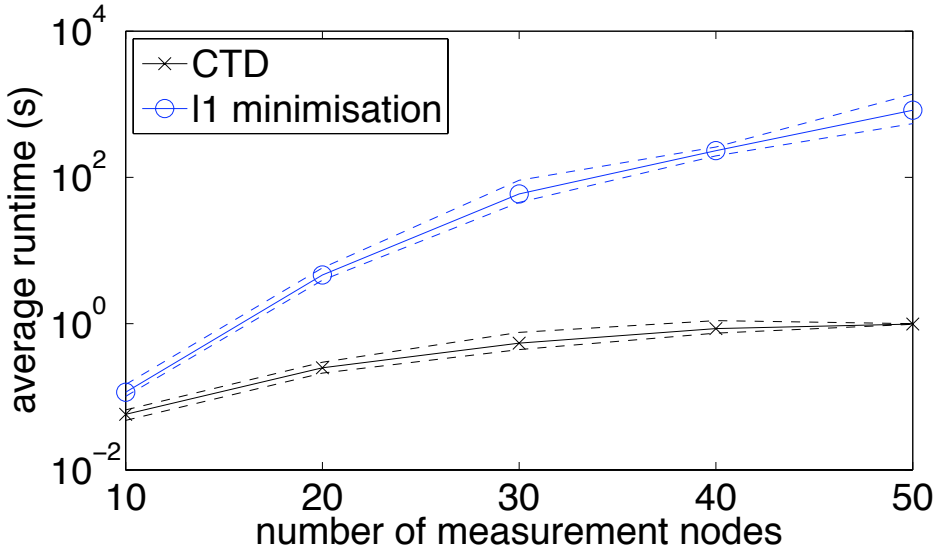


Figure 4.2: Logarithms of the run times as the size of the problem grows, comparing the two estimation methods. These are runtimes with Measurement Style 1 on Scenario 1 topologies (see Chapter 5), but the times are very similar when using different measurements. When scenarios with greater numbers of links are used (*e.g.,* Scenario 2) then the times are greater, but their relative magnitudes are almost the same as above. The dashes indicate bootstrapped 95% confidence intervals.

4.9 Conclusion

We show here that compressive sensing ideas cannot be immediately applied to link tomography on general networks, despite being intuitively appealing. Routing matrices poorly satisfy the typical conditions required for compressive sensing theorems in general, and the sparsest solution is not necessarily unique in a Tomographic problem, even when only estimating *logical* links.

Despite this, compressive sensing does work to a reasonable degree, but we can do better by actually exploiting the structure in routing matrices. We present an efficient algorithm: the first stage determines the solution on some set of links for which we have sufficient information, then the second stage applies a compressive sensing algorithm to find an accurate approximation to the sparsest solution on the remainder of the links. We find that this algorithm, CTD, is both faster and gives more information than directly applying compressive sensing. In the next chapter we test CTD on simulated topologies and find that CTD can *determine* a high proportion of the logical links, and only estimates incorrectly a small number of those it does not determine. Furthermore CTD exploits the highly-structured nature of routing matrices to be orders of magnitude faster than naïvely applying a standard compressive sensing algorithm, ℓ_1-minimisation.

Chapter 5

The effect of network topology on Coherent Tomographic Deduction

5.1 Introduction

Chapter 4 elucidated the multiple source-multiple destination link tomography problem and provided an algorithm for addressing it: Coherent Tomographic Deduction (CTD). This raises the question: How do we test this algorithm? Chapter 3 demonstrated a means of synthesizing topologies. We can use these topologies to test CTD, and that is the theme of this chapter.

There are two big benefits to using these synthesized topologies to test CTD. The first benefit is we can generate a sufficiently large number of them to get statistically reliable results for our algorithm. The second benefit is that the method in Chapter 3 can generate different "styles" of topology by changing the optimisation parameters k_0, k_1, k_2, k_3. These different styles represent the differing forms network topologies can take when they are formed in differing real-world circumstances. This allows us to test CTD on a wide range of topologies, so we can determine how the underlying performance objectives used when constructing a network can affect the ability to successfully perform tomography on that network. Additionally, the wide range of input topologies allow us to more thoroughly test CTD and gain confidence in our conclusions. In this chapter we see that CTD solves the link tomography problem very successfully, and is limited more by fundamental limitations in the measurements than by its own shortcomings.

5.2 Test methodology

To test CTD, we apply it to a range of synthesized topologies. Applying CTD is a multi-stage process, and in order to understand where the real gains are occuring, we break the results down into how many unidirectional links were determined, *i.e.,* solved or estimated, at each step. Before we can apply CTD though, we have to establish the tomography problem to which it will be applied.

We start by synthesizing topologies on 50 nodes with three different parameter sets, i.e. three different tuples of k_0, k_1, k_2, k_3 values. These tuples are shown in Figure 5.1. Each parameter set represents a different level of "hubbiness" and a different level of "meshiness" - that is, differing numbers of degree one nodes and differing numbers of links, respectively. The higher the value of k_2, the more meshy the network, i.e. the more links the network has. The higher the value of k_3, the fewer the number of hub nodes and the greater the number of leaf nodes; where leaf nodes are those with only one connection to other nodes, and hub nodes are those with more than one connection to other nodes. We choose these three particular tuples of parameters to present here because they produce very distinct types of networks while also each approximating many of the networks found in the Internet Topology Zoo.

Scenario 1 has $(k_2, k_3) = (0.001, 100)$. This produces topologies with few hubs and many leaves because k_3 is relatively large. Scenario 2 has $(k_2, k_3) = (0.0001, 10)$, giving networks with low average node degree and low coefficient of variation of node degree (CVND) because most nodes are hubs. Scenario 3 has $(k_2, k_3) = (0.01, 1)$, resulting in topologies with high average node degree. This information is summarized in Table 5.1.

Scenario	k_2	k_3	Description
1	0.001	100	Hub and leaf style topology
2	0.0001	10	Low node degree
3	0.01	1	High node degree

Table 5.1: The topology synthesis scenarios used for testing CTD.

Once we have determined a topology, we allocate loss to links randomly. For these experiments each link, i, has a 15% chance of having loss on it (transmission rate less than 1). If a link has loss on it then its transmission rate τ_i is uniformly distributed between 90% and 99%.

Now we must choose which paths in that network to measure. The first step is to choose which nodes can perform measurements. We pick a number of nodes (10,20,30,40,50) to perform measurements, and then allocate these measurement centres randomly to nodes of the synthesized network.

Next we choose what style of measurements we are going to take. Measurement Style 1 is simply one-way measurements from each measurement node to each other measurement node. Measurement Style 2 are round trip measurements, one from each measurement node to each other node and back. Measurement Style 3 has all the measurements from Measurement Styles 1 and 2 combined. For example, say we have a 50 node network with 20 measurement nodes. Then Measurement Style 1 will result in 20×19 measurements. Measurement Style 2 will result in 20×49 measurements. Measurement Style 3 will result in $20 \times (49 + 19)$ measurements, since the one-way source-to-source measurements from Measurement Style 1 are different from the round trip ones of Measurement Style 2, even when the end points are the same.

Since we only measure some of the paths in the network, we might not be measuring all the links in the network. The only links for which we can hope to determine loss rates are those that lie on paths that we measure. Furthermore, any set of links that appears on exactly the same paths will be indistinguishable in terms of our measurements. Consequently, we call these links redundant and apply a standard tomographic technique of aggregating them into "logical links".

In summary, to determine a problem we:

- Choose k_0, k_1, k_2, k_3 from one of three scenarios and synthesize a 50 node topology.
- Allocate loss on the links.
- Choose $(10, 20, 30, 40, 50)$ nodes as measurement nodes (or *sources*).
- Choose measurement style $(1, 2, 3)$ - this determines the paths we are measuring and the logical links.

5.3 Performance metrics

Once we have determined the tomography problem, we can apply CTD. In order to examine the effectiveness of CTD we can look at a breakdown the number of links determined at each step (or estimated, depending on the step), and how this changes in relation to the type of network and the choice of measurement infrastructure. We examine how effective the measurements are at sensing links in the network, and how effective CTD is at determining/estimating the loss values for links once measurement has taken place. While the measurements are not always effective, CTD always uses them efficiently to determine the loss values on a very high proportion of links.

We break the processes of measurement and estimation down into separate steps so we can examine how each step affects the overall solution.

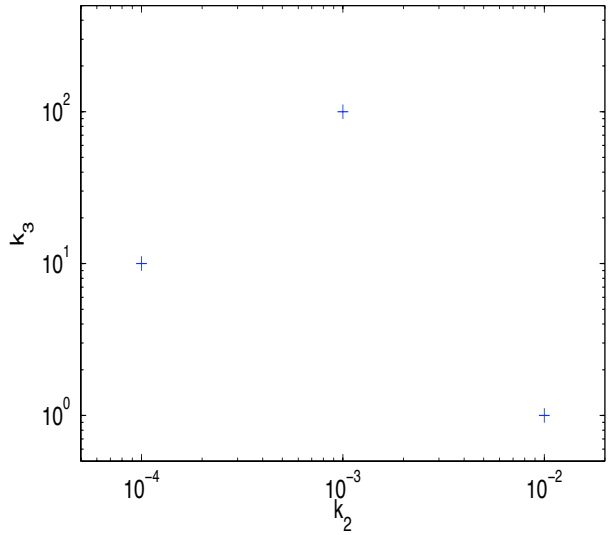


Figure 5.1: The k_2, k_3 values for each of the three styles of topology, or "scenarios". The higher k_2 is, the more links there are in the resulting topology. For a given value of k_2, increasing k_3 increases the number of degree one nodes (leaves); and reduces the number of nodes with degree greater than 1 (hubs).

First, we only measure some of the paths through the network. Not all links will lie on a measured path. Any link that lies on a measured path is called a *measured link*.

Next, some of these links are redundant and get aggregated into logical links. We display both of these cases on the plots that follow. Note that no tomography algorithm can hope to determine any more than the logical links - at this stage only our choice of network and measurements will affect how many logical links remain. The logical links remaining after the first two steps are the only ones we can hope to determine regardless of the tomography algorithm we choose to use.

Now we apply CTD. However, in order to see if CTD gives any real benefits we compare it to a trivial algorithm that does two simple steps:

- Determines all links that are on paths with no loss as having no loss, and removes those links from the problem.
- Then determines all links that are on single link paths and removes them from the problem.

Due to the structure of CTD, it will always be at least as successful as these two steps. However, it is important to see what advantage it has over those simple steps, and whether every stage of CTD is necessary.

Finally, we apply the two stages of CTD - the determination/reduction stage and the

estimation stage (ℓ_1-minimisation). For stage 1 we display how many links are left *undetermined* after its application. For stage 2, we display how many links it *estimated* incorrectly. Note that for each of the following graphs, the number of links remaining unsolved is expressed as a proportion of the total number of links in the network.

At each stage of the process, some of the links are removed from the problem; either because they are impossible to resolve (unmeasured links and indistinguishable links) or because their link-loss values have been determined. The lines on the graph are always in the same vertical order, since they represent the number of links remaining in the problem after each of the preceding steps. These steps are as follows.

- Measuring along paths. The remaining links are measured links - the upper black line  .
- Aggregating redundant links into logical links. The remaining links are logical links - the upper blue line  .
- Determining loss rates for links on paths with no loss. The remaining links are those 'unsolved after no loss paths' - the red line  .
- Determining loss rates for links that are measured directly. The remaining links are those 'unsolved after single link paths' - the green line  .
- Reduction/determination stage of CTD. The remaining links are 'unsolved after CTD stage 1'- the lower black line  .
- The estimation stage of CTD, utilizing ℓ_1-minimisation. The remaining links are 'incorrect after CTD stage 2' - the lower blue line  .

The areas we would like to be as small as possible are the area above the upper blue line and the area below the lower blue line. The area above the upper blue line represents flaws in our measurement infrastructure – regardless of the tomographic method used. The area below the lower blue line represents links estimated incorrectly by CTD. We find that there are usually many more links unavailable to the measurement infrastructure than links that are measured but cannot have their values resolved by CTD.

5.4 Results

We have nine main figures, one for each combination of topology scenario (Table 5.1) and measurement style. Each figure is split into two parts : Part (a) which displays all

the curves and Part (b) which is a magnified view of the lower section of Part (a) in order to more easily show the results of applying CTD.

The data plotted are the means for each of a series of experiments: each point on a solid line is the mean of fifty experiments, ten on each of five sample topologies. Every solid line has dashed lines surrounding it. These are the bootstrapped 95% confidence intervals for the mean [81].

We start by explaining in more detail what each of the curves represents, and highlight the trends common to all the graphs.

5.4.1 Measured and Logical Links

The uppermost black line in each graph represents the number of links that are measured. This is determined by the network itself, the choice of measurement nodes, and the measurement style. The solution algorithm (CTD) and the location of links with loss on them have no effect on the number of measured (or logical) links.

With Measurement Style 1, measurements are only taken along paths from one measurement node to another measurement node. Naturally, the greater the number of measurement nodes, the greater the number of links on measured paths. There are 50 nodes in the network in total, so as the number of measurement nodes tends to 50, the proportion of links measured tends to 100%. With 50 measurement nodes, every link is on a path. If there is a link from node A to B then that link will be on the path from node A to B, since (under the Euclidean distance metric) the shortest path is the most direct one. Furthermore, the path from A to B will *only* be the link from A to B, so the measurements will be capable of distinguishing every link from every other link; that is, there is no aggregation into logical links - there are the same number of logical links as physical links.

Measurement Style 2 measures links via round trip paths from the measurement nodes to every node. Since links are unidirectional, a link (A,B) will always be on the same paths as the link (B,A). This means that while Measurement Style 2 can measure nearly all the links, the pairs of unidirectional links (A,B) and (B,A) will always be indistinguishable, and hence incorporated into one logical link. Thus, with Measurement Style 2 there will always be 50% or fewer logical links than measured links.

Just as in Measurement Style 1, in Measurement Style 3 the proportion of measured links (and logical links) increases to 100% as the number of measurement nodes increases to 50, because Measurement Style 3 incorporates all the measurements of Measurement Style 1.

The main point to note is that even with a fairly large number of measurement nodes and a very large number of paths through the network, some links are still not being measured in simple source-to-destination measurements. Using round-trip measurements, *e.g.,* pings, far more links are measured, although unfortunately pairs of links between the same nodes (but in different directions) can no longer be disambiguated. In measurement style 3 we get the best of both worlds, although with correspondingly more measurements.

5.4.2 Links on no loss paths

The red line on each graph represents the number of links after the first and simplest step in CTD: no loss paths. The *no loss paths* step considers all the paths with no loss on them, and concludes that any link on any of these paths must have no loss. These links are considered resolved and removed from the problem. The paths with no loss on them are removed from the problem also, since they have no more information to provide.

Links remaining after this step either have loss on them, or are only visible on paths where other links have loss. Since roughly 15% of all physical links have loss on them, the number of links unsolved after the no-loss-paths step is at least 15% of the number of remaining *logical links* - all the lossy links plus some of the other links.

5.4.3 Unsolved after single link paths

Once links on paths without loss have been resolved, they can be removed from the equations. The next step is to resolve loss values for links on single-link paths. This utilizes paths that have only one link on them, either because they had only one logical link to start with, or because all the other links were resolved in the no-loss paths step.

The single-link paths and no-loss paths steps are essentially the baseline for the performance of the CTD algorithm: they pick all of the "low hanging fruit" - the links that are easy to resolve from the measurements. We see that, a lot of the time, these two steps are very effective, resolving 80% or more of the remaining links.

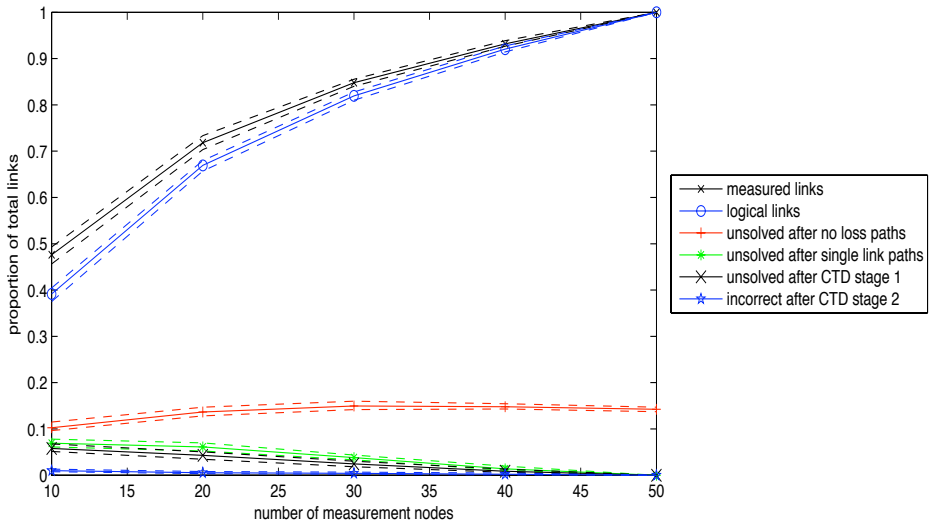
5.4.4 CTD results

The lowest four lines on each plot in Figures 5.2–5.10 represent the links remaining unsolved after each part of CTD. The lower each of those lines compared to the one above it, the more effective that part of the algorithm. Note that the black line represents

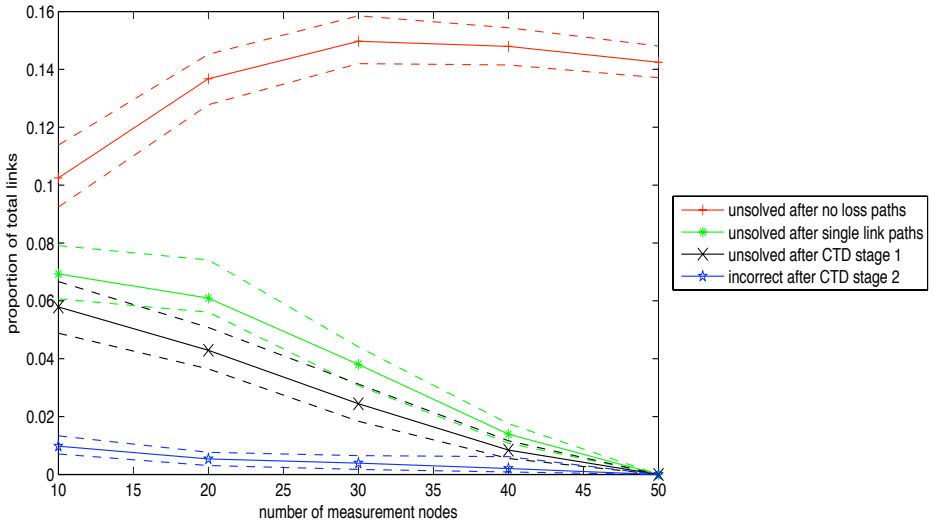
the links unable to be determined after CTD stage 1 - the remaining *unknown* links, whereas the blue line represents the error rate, or those of the remaining links that are incorrectly estimated by CTD stage 2. Links are considered correctly estimated if the estimated value is within 1 percentage point of the real loss rate. Detailed comments are made in the captions of each figure.

In general, the number of links left undetermined after CTD stage 1 is very low, and those incorrectly estimated by CTD stage 2 is even lower. CTD stage 1 and 2 typically leave the fewest logical links undetermined or incorrectly estimated when Measurement Style 2 is being used; however this is likely because Measurement Style 2 often resolves the fewest logical links.

Perhaps unsurprisingly, the performance of both stages of CTD is better the more measurement nodes there are – even though the number of logical links increases with increasing numbers of measurement nodes. In contrast, the performance of the no-loss paths step is not always improved by increasing the number of measurement nodes.

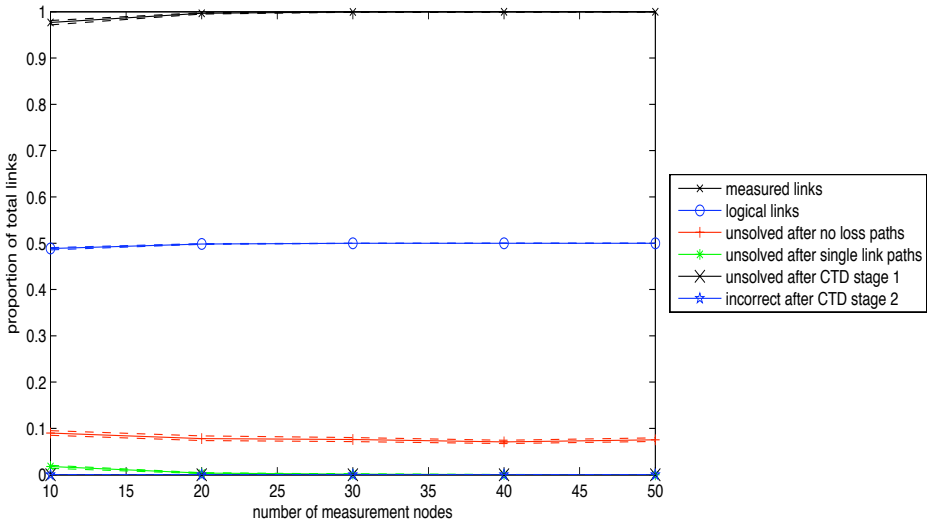


(a)

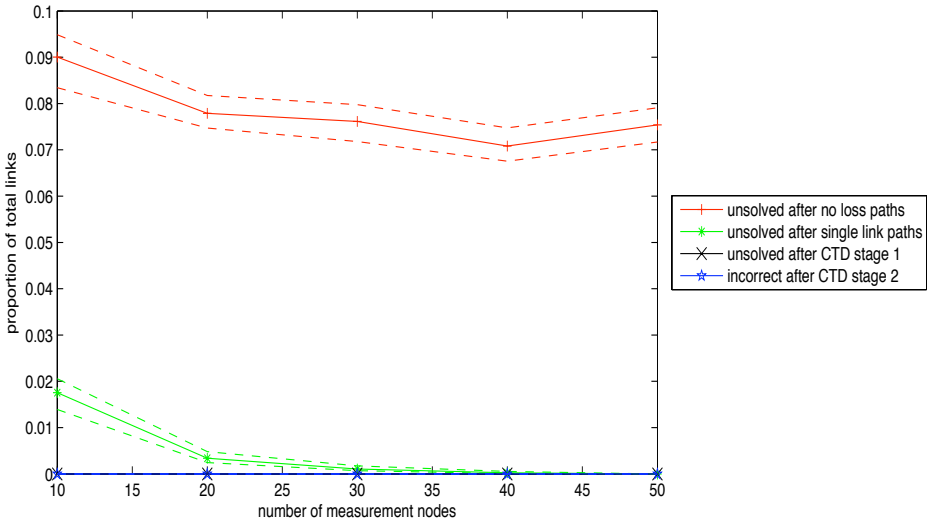


(b) magnified view of Figure 5.2a

Figure 5.2: *Topology Scenario 1*: In this scenario, there are 52 to 55 links and 6 to 8 hubs. *Measurement Style 1*: Measurements are only taken along paths from one measurement node to another measurement node. Even in the worst case, less than 2% of the links are incorrectly estimated by CTD, and only 6% are not determined by stage 1 of CTD. Since there is a notable decrease between each of the lines (except with every node a measurement node) we see that every step of CTD has a role to play.

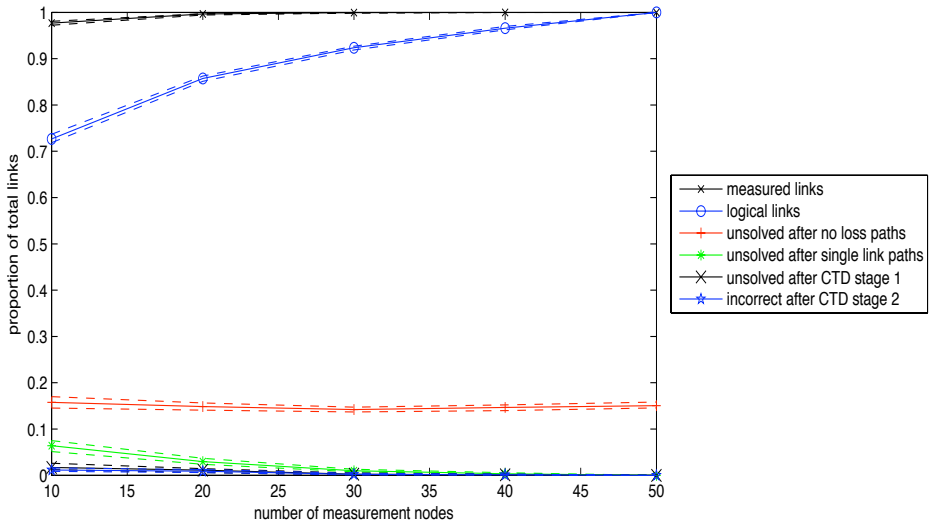


(a)

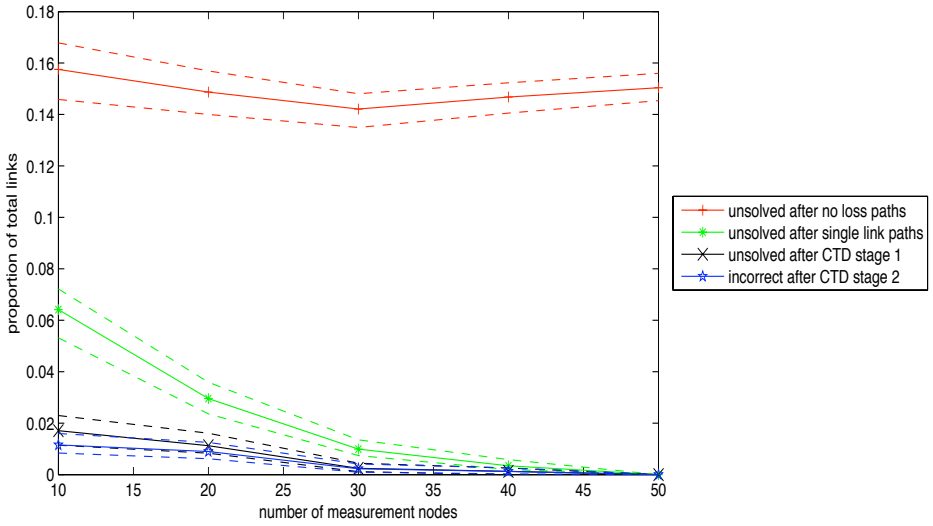


(b) magnified view of Figure 5.3a

Figure 5.3: *Topology Scenario 1, Measurement Style 2*: *Round-trip* measurements are taken from every measurement node to every node. We see that in this scenario, with round trips from the measurement nodes to every node, almost all links lie on a measured path. Measurement Style 2 then causes the unidirectional links to be aggregated into a bidirectional logical link for the reasons detailed above. This means that the logical links (blue) will be at most 50% of the measured links (black). The performance of CTD is outstanding in this case, with less than 0.1% of links remaining undetermined after CTD stage 1.

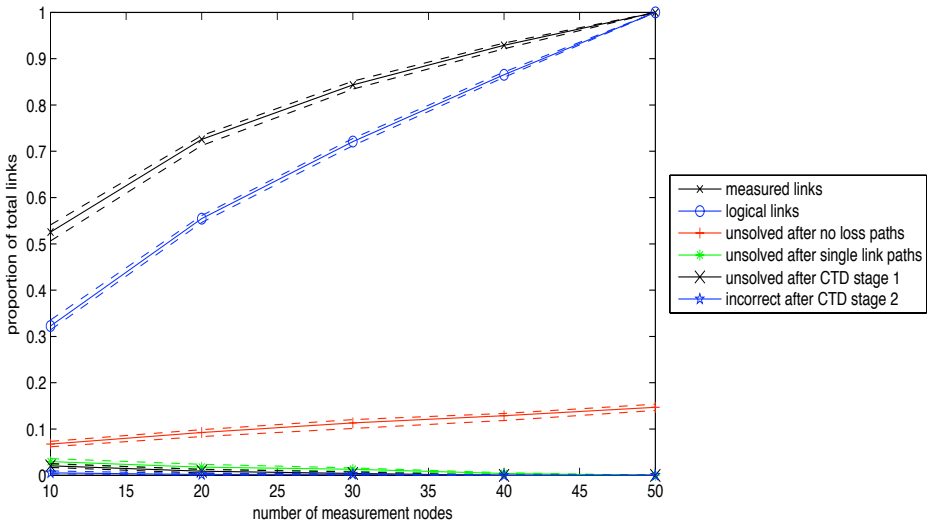


(a)

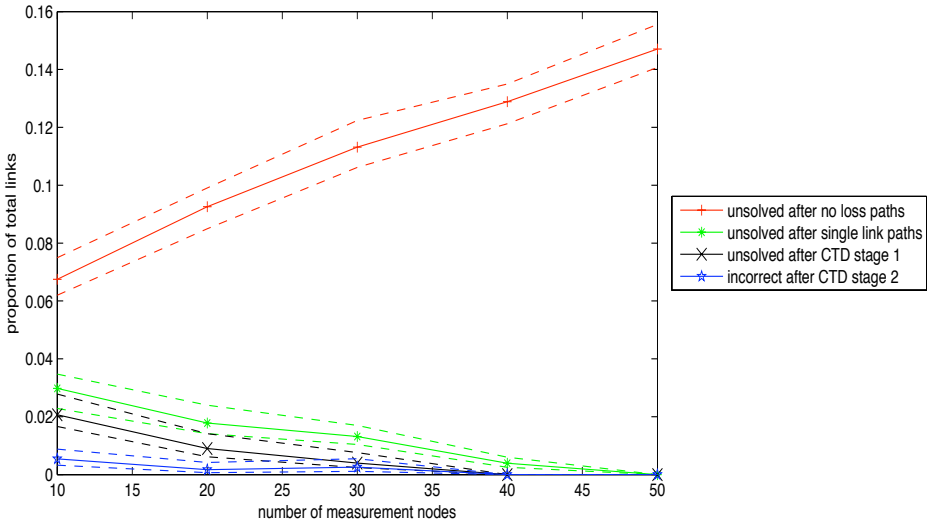


(b) magnified view of Figure 5.4a

Figure 5.4: *Topology Scenario 1, Measurement Style 3*: both measurement styles are combined - both measurement-node-to-measurement-node measurements and round trip measurements to all nodes. Like Measurement Style 2, we have almost all links being measured even with only 10 measurement nodes. However, there is far less aggregation into logical links than in Measurement Style 2, showing that measurements from Measurement Style 1 allow the measurements to distinguish more than 20 percentage points extra links than in Measurement Style 2. While there is much greater aggregation of links happening than in Measurement Style 1, there are still overall many more logical links in Measurement Style 3 than in Measurement Style 1. Interestingly, even though the number of logical links is increasing with higher numbers of measurement nodes, the overall number of those unable to be resolved by no loss paths remains approximately constant. The additional information from more measurement nodes balances out the increase in logical links.

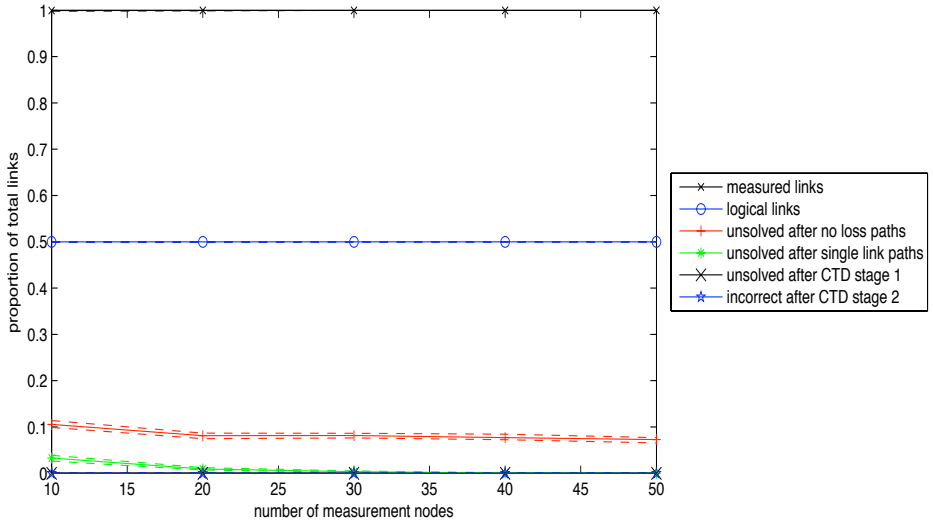


(a)

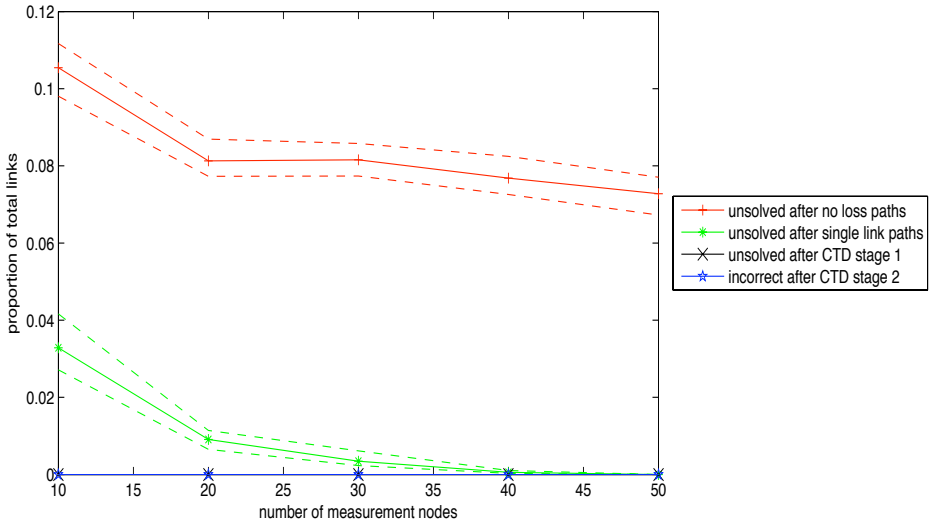


(b) magnified view of Figure 5.5a

Figure 5.5: *Topology Scenario 2*: This scenario has very few links (51 to 53), so the network is almost a tree. About half the nodes are leaf nodes and half the nodes are hub nodes. *Measurement Style 1*: Measurements are only taken along paths from one measurement node to another measurement node. Compared to the same measurement style for Scenario 1 (Figure 5.2), a greater proportion of links are measured links. However, many fewer of the measured links are able to be distinguished due to the tree-like structure of the networks, so there are fewer logical links. As the number of measurement nodes increases, the number of logical links grows rapidly, becoming a primary driver in the number of links unsolved after the no-loss-paths step. As we can see, CTD performs well in tree-like environments, even better than in Scenario 1. Note that there are fewer links and fewer logical links overall, and that the performance of any algorithm is capped by being unable to distinguish those measured links that are aggregated into logical links, so this scenario is not a desirable one overall.

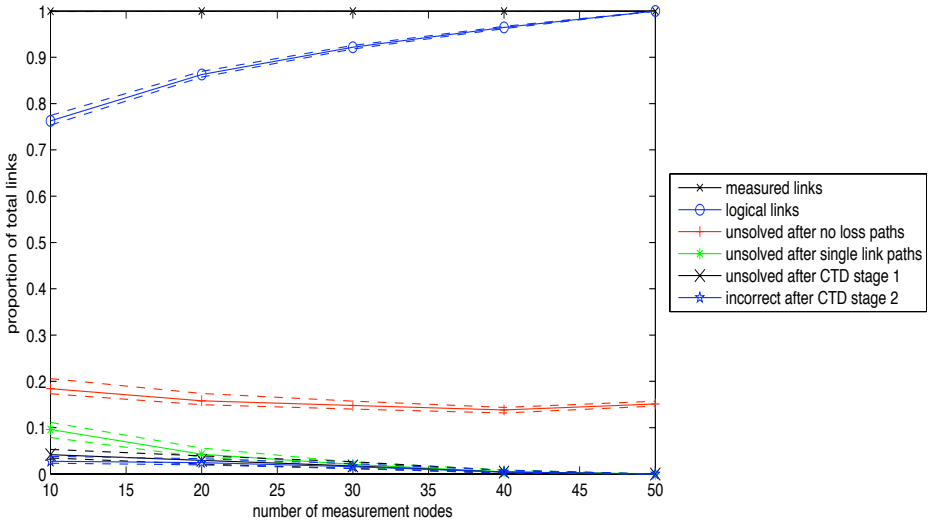


(a)

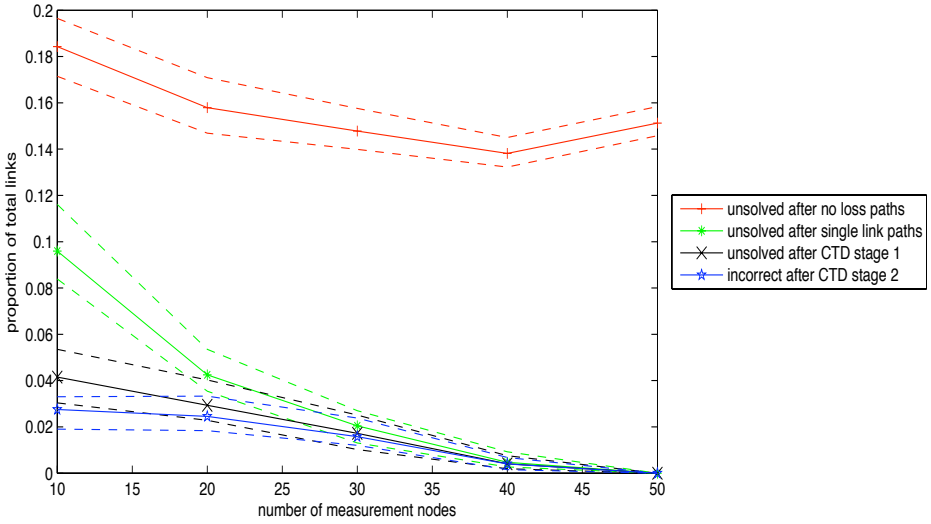


(b) magnified view of Figure 5.6a

Figure 5.6: *Topology Scenario 2, Measurement Style 2*: *round-trip* measurements are taken from every measurement node to every node. In this scenario, the network is almost a tree. Because of this, when measuring using round trips to every node then it is highly likely every link will be measured. We see that this is in fact the case in the plot: even with only ten measurement nodes, we have 100% of the links measured. As per usual with this measurement style, these measured links are aggregated in pairs into logical links, giving logical links equal in number to half the total physical links. In Measurement Style 2, logical links are often bidirectional aggregates of two coinciding unidirectional physical links. Consequently, if a logical link is experiencing loss then it is clear which bidirectional physical link has the problem, it's just unclear which direction the problem is occurring in. With increasing numbers of measurement nodes we have the same logical links, but better information about them, and this is reflected in the increased effectiveness of each step in the CTD method. Note that the links are all determined successfully by stage 1 of CTD, so stage 2 is unnecessary.

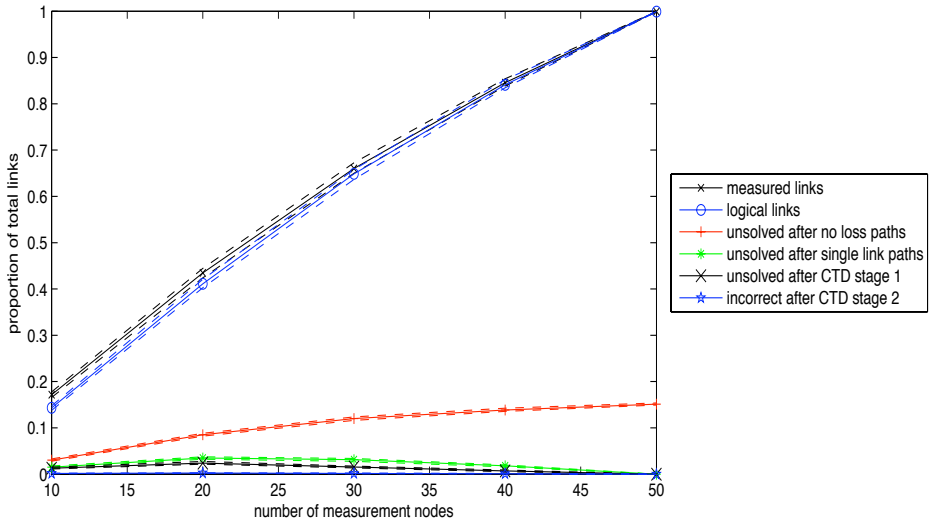


(a)

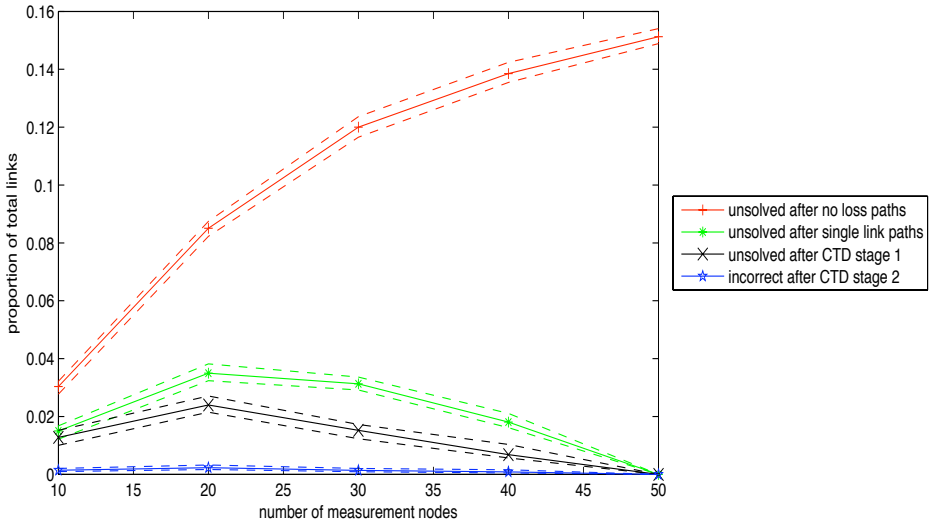


(b) magnified view of Figure 5.7a

Figure 5.7: *Topology Scenario 2, Measurement Style 3*: both measurement styles are combined. Here we have the combined measurements of Figures 5.5 and 5.6 applied to the tree-like topology. Interestingly, despite CTD correctly estimating all logical links with Measurement Style 2, when adding the unidirectional measurements of measurement style 1, the method can no longer correctly resolve all the logical links. The additional measurements serve to distinguish pairs of unidirectional links, but some of those newly distinguished links are now estimated incorrectly.

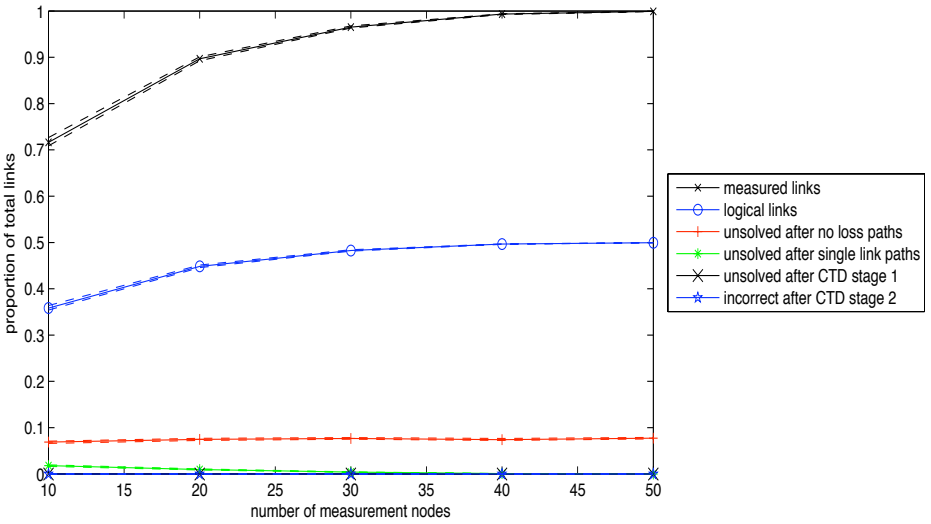


(a)

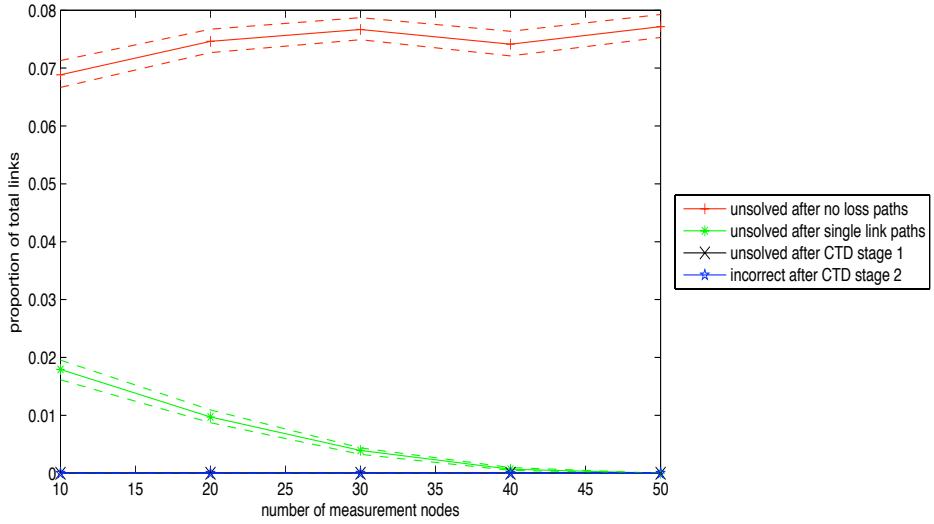


(b) magnified view of Figure 5.8a

Figure 5.8: *Topology Scenario 3*: Scenario 3 has networks with roughly 280 links. There are so many links that typically all nodes are hubs. Measurement Style 1: Measurements are only taken along paths from one measurement node to another measurement node. Because these networks have so many links, the measurements are ineffective at finding all of them. However, as we can see from the number of logical links, the measurements used *are* effective at distinguishing the links they do find. Unlike previous scenarios, CTD stage 1 is actually more effective with 10 measurement nodes than 20 measurement nodes, for the links it can actually measure and distinguish. CTD stage 2 is *very* effective at determining all the links, with a less than 0.2% error rate for every number of measurement nodes.

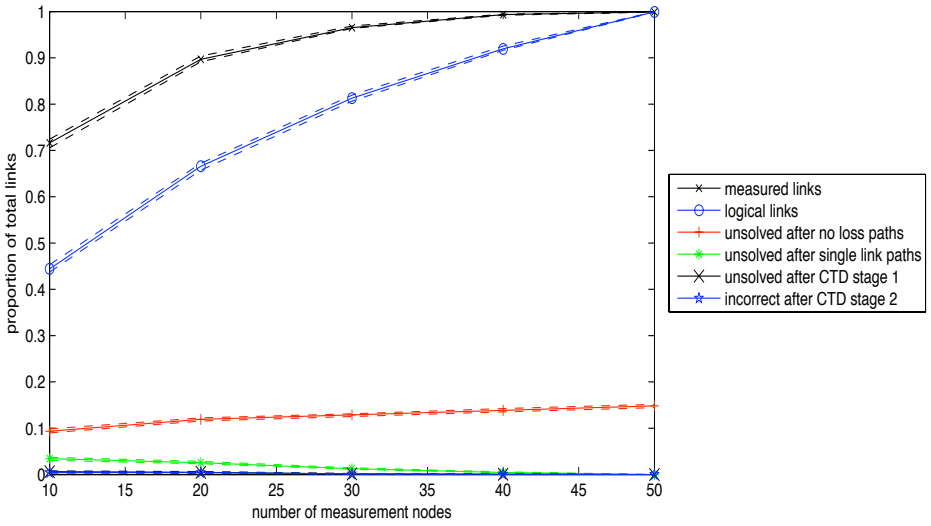


(a)

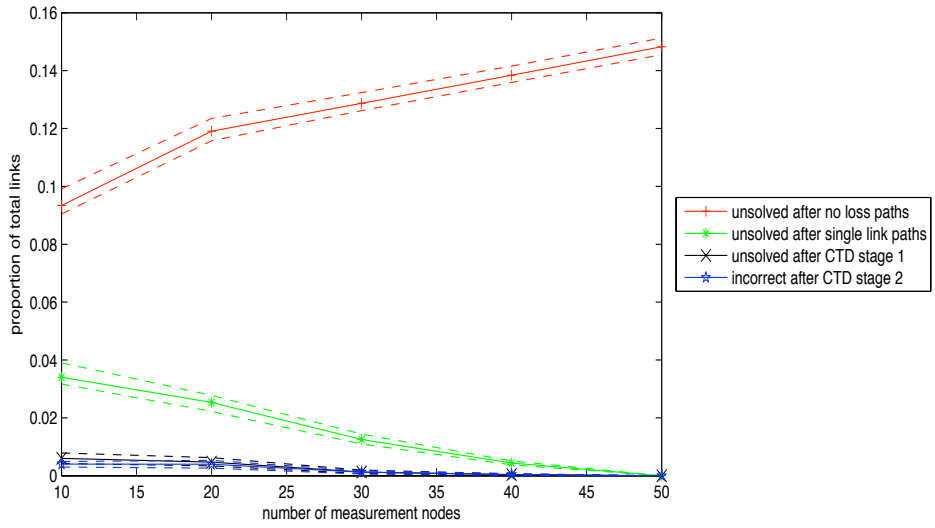


(b) magnified view of Figure 5.9a

Figure 5.9: *Topology Scenario 3, Measurement Style 2*: *round-trip* measurements are taken from every measurement node to every node. For the previous two scenarios, Measurement Style 2 can measure all or nearly all of the links. However, in this case, there are so many links that even Measurement Style 2 measures less than 75% of them for 10 measurement nodes. As per previous scenarios with Measurement Style 2, we still have 50% of those links as logical links. In this case, CTD stage 1 once again functions perfectly and we can successfully solve for all of these logical links.



(a)



(b) magnified view of Figure 5.10a

Figure 5.10: *Topology Scenario 3, Measurement Style 3*: both measurement styles are combined. Much like in previous scenarios, measurement style 3 combines the good coverage of measurement style 2 with measurement style 1's ability to disambiguate links. The topology induced by the measurements is amenable to CTD, with fewer than 1% of links estimated incorrectly.

5.5 How effective are our measurements?

In this section we answer the questions: How effective is each measurement style? How does the choice of topology affect how well the measurements work? Rather than plotting against the number of nodes, we take a different perspective by using k_2 and k_3 as the independent variables in this case. We see how the number of logical links sensed is affected by the measurement style. We discover the interesting result that the proportion of links visible to the sensing apparatus is not always monotonic in k_2: increasing the number of links in a network (by increasing k_2) initially makes it possible to sense a higher proportion of the links in the network, before the large number of links eventually overwhelm the sensing apparatus.

We also test how the effectiveness of CTD varies when moving through the space of topologies, and how measurements interact with this. While Measurement Style 2 is frequently the worst for distinguishing physical links (it provides the fewest logical links), it proves to be the best at allowing CTD to determine the loss values for those logical links. We also find CTD to be effective regardless of choice of measurement style 1 or 3, or to the type of topology (*i.e.,* changing values of k_2 and k_3 – at least over this range). See Figures 5.11–5.14 and their captions for more details.

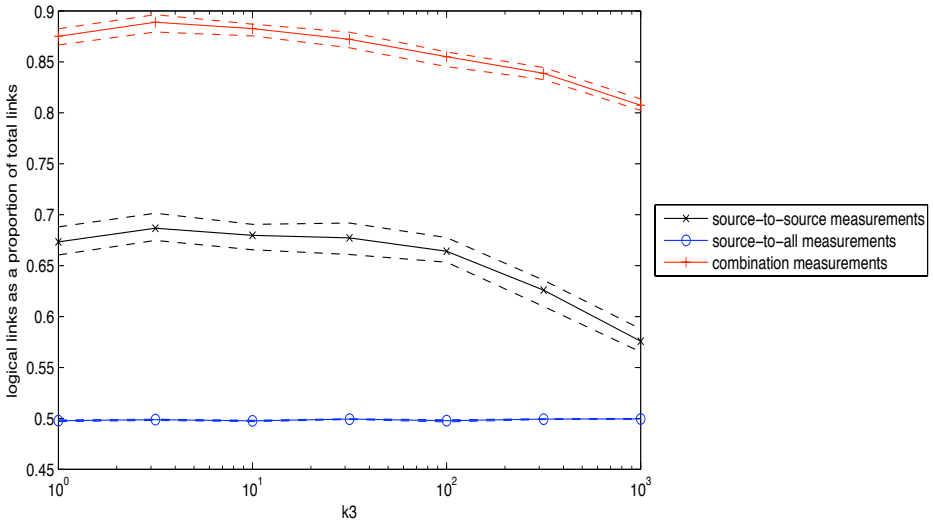


Figure 5.11: Plot of the number of logical links as a proportion of total physical links (the upper blue curve on the previous graphs) versus k_3. k_2 is fixed at 10^{-3} as per Scenario 1, and there are 20 measurement nodes. Different curves on the figure are different measurement styles. This figure allows us to directly compare measurement styles across changing topologies. The proportion of logical links is a measure of the effectiveness of each measurement style to measure and distinguish physical links. As the value of k_3 (and hence the "hubbiness" of the network) increases we see that Measurement Style 1 becomes decreasingly effective for measuring and distinguishing all the links in the network. However, Measurement Style 2 is unaffected by the hubbiness of the network increasing. Despite this, Measurement Style 1 is still more effective than the Measurement Style 2. The combination of both styles of measurements decreases efficacy when the source-to-source measurements are less effective. Even in the best case, more than 10% of the links are unmeasured or unable to be disambiguated from other links. So we see that even with 40% of nodes dedicated to making measurements, part of the network is always likely to be "invisible".

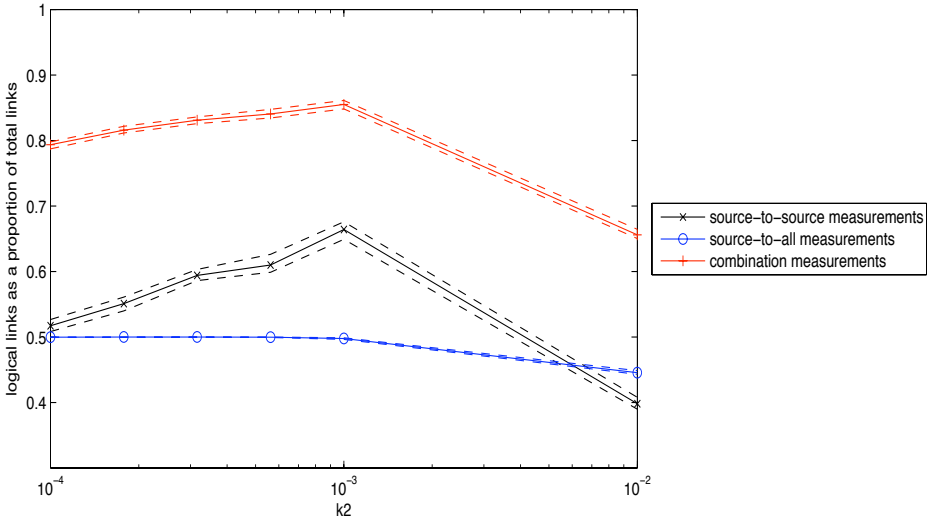


Figure 5.12: Like Figure 5.11 this figure compares the number of logical links (as a proportion of total links) resulting from different measurement styles – but this time versus k_2. k_3 is fixed at 100, the same value as in Scenario 1. This allows us to compare the effectiveness of each form of measurement as the number of links in the graph increases (with increasing k_2). Interestingly, the curves are not all monotonic – initially the increasing number of links in the core of the network allows a greater proportion of them to be successfully measured and distinguished, but then there are so many links that many of them are no longer on paths between measurement nodes. Note that once again, fewer than 90% of the total links appear as logical links.

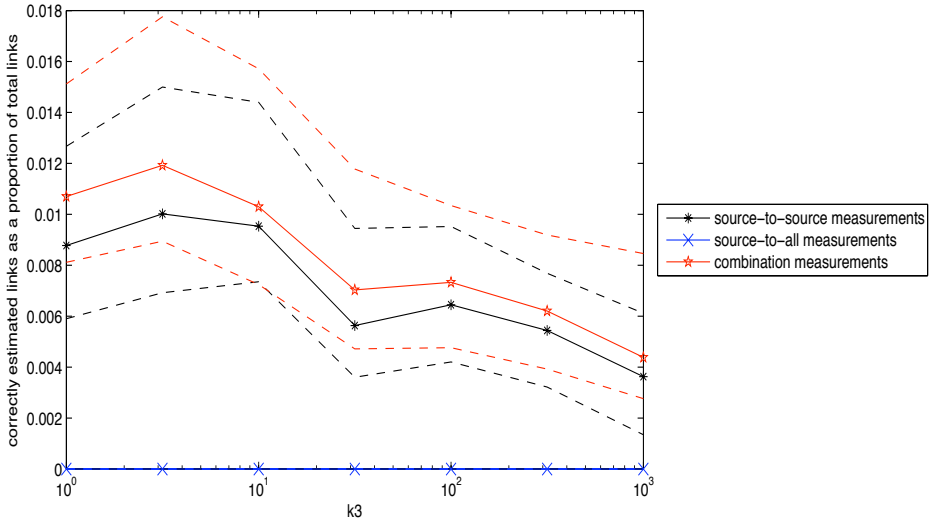
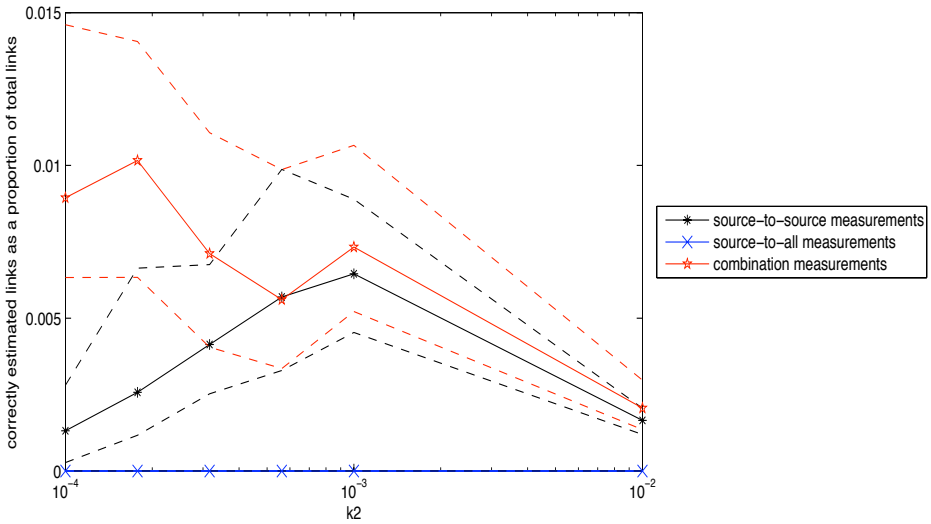


Figure 5.13: Plot of the number of logical links that CTD *incorrectly* estimates as a proportion of the total number of physical links versus k_3. Here we vary k_3 and examine how each measurement style changes the effectiveness of CTD. k_2 is fixed at 10^{-3} as per Scenario 1, and there are 20 measurement nodes. Note that all the links are correctly resolved for Measurement Style 2 although there are fewer logical links to determine, as per Figure 5.11. CTD is still very effective for Measurement Styles 1 and 3 regardless of k_3, although there could be a slight downward trend in the number of errors as hubbiness increases, perhaps caused by the reduced number of logical links that are measured.



(a) $k_3 = 100$

Figure 5.14: The number of links incorrectly estimated as a proportion of total links (like Figure 5.13), this time plotted against a changing value of k_2 used in the topology generation algorithm. Increasing k_2 causes an increase in the number of links and a decrease in the average path length between measurement nodes. The relative uncertainty for these curves is high because the absolute values of these curves are very low relative to the actual number of links. CTD is very effective at recovering the link-loss values for each style of measurement – and in the case of Measurement Style 2 was always successful, although this measurement style has the fewest logical links sensed.

5.6 Conclusion

In general, if measurement nodes are placed randomly, even the measurements made from a relatively large number of measurement nodes are surprisingly poor at covering the links. Within this restriction, CTD is extremely effective at determining the link loss rate across a wide variety of networks, usually estimating fewer than 5% of the total link loss rates incorrectly, and regularly performing much better than that.

Up until now, we have only considered the idealized situation with no measurement noise at all, where loss rates on paths can be estimated arbitrarily accurately. While this is a frequent assumption, it can place substantial restrictions on the application of this work. In the next chapter, we address this assumption and show a small alteration to our method that allows accurate estimation of link loss values in environments with measurement noise imposed by the probing process.

Chapter 6

Link loss tomography with measurement noise

6.1 Introduction

Chapter 4 introduced a method for performing link loss tomography on general topologies, Coherent Tomographic Deduction (CTD). In Chapter 5 we see that CTD is very effective at finding link loss rates, even with very limited information from measurements. One barrier to directly applying this tomographic work is that, in practice, we are not supplied with exact loss rates along paths. Measuring loss rates by probing the network is a stochastic process and only gives approximate loss rates. Even a simple model of probing must allow for some stochastic variation in the measured outputs.

The simplest and most well known stochastic model for probing is the following: Start by assuming that each time a probe traverses a link it has an independent chance of being lost, fixed for that link. Let t_i be the probability of it being transmitted successfully through link i were it to reach link i. Then, each probe along a path q_j has an independent chance of being lost, p_j. Then the number of measured losses is a binomially distributed random variable $Y_j \sim \text{Bin}(n, p_j)$, where n is the number of probes sent along the path. The maximum likelihood estimate of the average loss rate along q_j is then $\hat{p_j} = Y_j/n$. However, generating a confidence interval for the estimate of p_j has some subtleties. Inverting the naïve Gaussian approximation to the Binomial has long been known to produce wildly varying coverage probabilities[1], even for values of n and p that appear safe from standard rules of thumb[2] [82]. Even the "exact" (Clopper-Pearson) confidence interval (the one used by default in Matlab) has

[1]Coverage probabilities are the probability that for a given set of parameters that those parameters are inside the generated confidence intervals.

[2]For example, $np, n(1-p) > 10$.

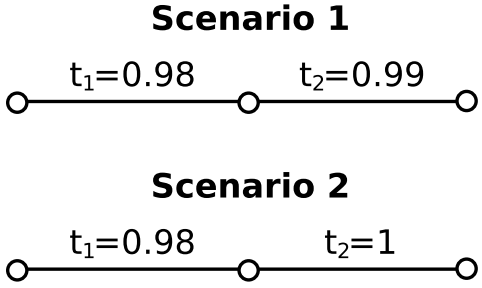


Figure 6.1: An example to demonstrate the difficulty of determining link loss rates using probes. The variance from the loss rate on link 1 obscures losses on link 2, even when we know the loss rate on link 1 perfectly.

undesirable coverage probabilities: it is very conservative and often has a much higher coverage than is required [82,83]. There are a variety of reasonable confidence intervals that can be used instead, here we use the Wilson interval for its relative simplicity and well-behaved coverage probabilities [5,83].

In network link tomography it's frequently assumed that we can get arbitrarily accurate estimates for parameters like loss and delay, but in practice this can require a very large number of probes. The following simple tomographic example demonstrates this.

Consider two links, link 1 and link 2 and a short path, q consisting of both these links and no others. Let t_i be the transmission rate on link i. Assume for the sake of the example we know $t_1 = 0.98$. We wish to distinguish between two possible scenarios using measurements along q: either $t_2 = 0.99$; or $t_2 = 1$ (Figure 6.1). If we take 1 measurement per second for 10 minutes, this gives 600 measurements. Under the probing and loss model above, these two scenarios will be very difficult to distinguish reliably. Consider a hypothetical experiment of sending 600 probes through q_1 in scenario 1 and 2. In more than 11% of these experiments, the path in scenario 2 will have *more* losses, even though it has a lower actual loss rate. This means that when attempting to distinguish between these scenarios (using just one set of path measurements through link 2) that *any* method will get it wrong *at least* 11% of the time.

The previous example is characteristic of the first major challenge when measurement noise is included in link-loss tomography. Uncertainty in the loss rates along paths leads to inability to determine whether an individual link is suffering from *any* loss at all. In particular, links or paths with higher loss rates will have a higher *variance* in the number of loss events occurring along them. This high variance in loss rate can mask the loss occurring on other links, especially ones with lower loss rates (as in the example above). The fact that loss on a link might not be discernible in the variation due to measurement noise only adds to the other topographic identifiability problems detailed in previous chapters.

The second major difficulty when introducing stochastic measurements to the link-loss tomography problem is choosing an appropriate model for loss and loss events on probes. In this work, we stick to the model of loss described above: each probe individually has a probability of failure equal to the loss rate on the path along which it is sent – loss events occur on a link independent of loss events for other probes or on other links. We keep the loss model simple to make evaluation of our tomography algorithm easier.

CTD, as defined in Chapter 4, will not adequately deal with stochastic measurements, since several steps require exact (or very accurate) loss rates to succeeed. Consequently, we again turn to compressive sensing techniques for inspiration and justification. This time, we deal with compressive sensing in the presence of noise.

6.2 Compressive sensing in the presence of noise

All the compressive sensing we have considered up until now (primarily in Chapters 2 and 4) has been in the case of measurements without errors or noise. However, perhaps surprisingly, some of the same results apply with relatively small adjustments for noise.

In the exact case, covered in Chapter 2, the ideal sparse recovery problem to be solved by compressive sensing takes the following form:

$$\min_x ||\mathbf{x}||_0 \text{ subject to } \mathbf{y} = \Phi\mathbf{x}. \tag{6.2.1}$$

Let us now suppose that instead of being supplied with the exact measurements, $\mathbf{y} = \Phi\mathbf{x}$, we are supplied with a noisy version of the measurements, $\mathbf{y}_n = \Phi\mathbf{x} + \mathbf{e}$, where $\mathbf{e}$ is some error term. In the presence of noise, rather than finding the sparsest signal that exactly satisfies the measurements, we find the sparsest signal that *approximately* satisfies the measurements. Typically, this is represented by limiting the sum of squares distance between the measurements $\mathbf{y}_n$ and the measurement matrix Φ applied to the vector to be estimated, $\mathbf{x}$. The optimization problem is then:

$$\min_x ||\mathbf{x}||_0 \text{ subject to } ||\mathbf{y}_n - \Phi\mathbf{x}||_2 \leq \delta, \tag{6.2.2}$$

where δ is $||\mathbf{e}||_2$. However, as in the exact case, this problem is extremely computationally intensive [84], restricting us to using heuristics and approximations. Just as in the exact case, we can modify (6.2.2) by approximating the ℓ_0-norm with the ℓ_1-norm. This leaves us with

$$\min_x ||\mathbf{x}||_1 \text{ subject to } ||\mathbf{y}_n - \Phi\mathbf{x}||_2 \leq \delta. \tag{6.2.3}$$

We can further reformulate (6.2.3) to the convex optimization in Lagrangian form

$$\min_x ||\mathbf{x}||_1 + \lambda||\mathbf{y}_n - \Phi\mathbf{x}||_2^2, \tag{6.2.4}$$

which is equivalent to (6.2.3) for a suitable choice of $\lambda = \lambda(\mathbf{y}, \delta)$ [84].

The optimization formulations (6.2.3) and (6.2.4), termed ℓ_1-regularized least squares programs (LSPs [85]) have been investigated extensively for sparse approximation in compressive sensing, for example see [84, 86–88]. They have also been used in a variety of other areas, the most prominent being feature selection in statistics, where (6.2.4) is termed the Lasso [81, 89, 90].

Just as in the exact case, in compressive sensing with noise there are a variety of theorems and guarantees concerning the accuracy of the reconstructed signal. Many of these take the same form: given Φ and $\mathbf{y}_n = \Phi\mathbf{x}_0 + \mathbf{e}$, if Φ satisfies appropriate conditions[3] and $\mathbf{x}_0$ is sufficiently sparse, then the distance between the solution to (6.2.3) and the desired solution $\mathbf{x}_0$ is less than a constant factor times a norm of the noise $\mathbf{e}$, for instance $||\mathbf{e}||_2$ or $||\mathbf{e}||_1$.

For example, from [87]: Let $\mathbf{y} = \Phi\mathbf{x}_0 + \mathbf{e}$. If Φ obeys the uniform uncertainty principle[4], with unit columns, and $\mathbf{x}_0$ is sufficiently sparse, then the solution $\mathbf{x}^*$ to (6.2.3) is within the noise level

$$||\mathbf{x}^* - \mathbf{x}_0||_2 < C||\mathbf{e}||_2, \tag{6.2.5}$$

where C is some constant. We show how to apply this to link-loss tomography in the next section. Just as in the exact measurements case, routing matrices do not satisfy the criteria required of measurement matrices, but we can still apply these methods to our problem.

Solving (6.2.4) is not the only way to approach this problem. Padmanabhan *et al.* [70] approach this problem in three different ways. In the first way, links are given an ordering and loss assigned to links uniformly from within the feasible interval for that link. The second way is similar to solving (6.2.4), but instead they use the solution to

$$\min_x ||\mathbf{x}||_1 + \lambda||\mathbf{y}_n - \Phi\mathbf{x}||_1, \tag{6.2.6}$$

utilizing the ℓ_1-norm for the error term as well as the penalty term. This choice is reasonably successful but in Section 6.3 we explain why it is conceptually undesirable. Finally, they use Bayesian inference with a uniform prior and Gibbs sampling to sample from the posterior. This proves effective, but it is limited since it does not attempt to use sparsity in any way to make predictions.

[3] Usually the same conditions as in the exact case.

[4] The uniform uncertainty principle is essentially equivalent to RIP, the matrix criterion presented in Chapter 4.

6.3 Adapting ℓ_1-regularized least squares to the link-loss tomography problem

We start by defining the tomography with noise problem and relevant notation. We keep notation and assumptions the same as in Chapter 4, as far as possible. Let t_i be the transmission rate for edge e_i, and p_j be the transmission rate rate for path q_j. Losses on different links are independent, so

$$p_j = \prod_{e_i \in q_j} t_i, \quad \text{for all } j = 1, \ldots, m.$$

As previously defined, let

$$\begin{aligned} \tau_i &= -\log t_i, \quad i = 1, \ldots, n, \text{ and} \\ \rho_j &= -\log p_j, \quad j = 1, \ldots, m. \end{aligned}$$

With A as the routing matrix, we still have $\boldsymbol{\rho} = A\boldsymbol{\tau}$.

Now let the number of losses measured on each path be $Y_j \sim \text{Bin}(n, p_j)$, for $j = 1, \ldots, m$. From these measurements we will estimate t_i for $i = 1, \ldots, n$. The simplest and most direct way to do this would be to estimate p_i with the maximum likelihood estimate $\hat{p}_i = Y_i/n$, then calculate an estimate for ρ, $\hat{\rho}_i = -\log(\hat{p}_i)$ for all $i = 1, \ldots, m$. We would then apply (6.2.4) or (6.2.6), and solve

$$\min_{\boldsymbol{\tau}} ||\boldsymbol{\tau}||_1 + \lambda||\hat{\boldsymbol{\rho}} - A\boldsymbol{\tau}||_2^2, \tag{6.3.1}$$

or

$$\min_{\boldsymbol{\tau}} ||\boldsymbol{\tau}||_1 + \lambda||\hat{\boldsymbol{\rho}} - A\boldsymbol{\tau}||_1, \tag{6.3.2}$$

for $\boldsymbol{\tau}$ respectively.

As an aside, using (6.3.2) to find $\boldsymbol{\tau}$ is inappropriate since $||\hat{\boldsymbol{\rho}} - A\boldsymbol{\tau}||_1$ is inadequate in capturing the likelihood of a loss coefficient vector to produce the measurement vector. The problem is that the ℓ_1-norm effectively directly *sums* (in the log domain) all the differences between the expected loss rates for each path and the measured loss rates. However, increasing the difference for a path which is already highly different to the expected value represents a greater reduction in likelihood than increasing the difference for a path that is close to the expected value, because the measured loss rate is the (log of) a (normailsed) binomial variable. The ℓ_2-norm represents this situation much more accurately, but still not perfectly. It is a trade-off between ease and parsimony of the optimization on the one hand, and fidelity to the likelihood estimation on the other hand. The results in the next section show that it proves to be effective in finding the lossy links in simulations.

Matsuda *et al.* [91] use the optimization in (6.3.1) to make loss tomography estimates. They show reasonable success when applying it to measurements on an 8 node network with one randomly-chosen high-loss link. Unfortunately, directly applying (6.2.4) to give (6.3.1) is also inappropriate because it assigns equal weight to variations on each path; that is, equal weight in the cost function to each element of $\hat{\boldsymbol{\rho}} - A\boldsymbol{\tau}$. Consider $A\boldsymbol{\tau} = \rho$ and $\hat{\boldsymbol{\rho}} = -\log(Y_i/n)$. Since $Y_i \sim \text{Bin}(n, p_i)$, Y_i will have greater variance the closer $p_i = \exp(-\rho_i)$ is to $\frac{1}{2}$. This means that for values of p_i closer to $\frac{1}{2}$ we should expect a greater difference between $p_i = \exp(-\rho_i)$ and the value we are using to approximate it, Y_i/n. Similarly, for values of Y_i/n closer to $\frac{1}{2}$ we should expect greater differences between Y_i/n and p_i. This should in turn be reflected in the optimization process – differences between Y_i/n and p_i should be penalized less the greater the estimated variance of Y_i.

Changing the relative weights applied to measurement inconsistencies along different paths is complicated slightly by the fact that we are not using the direct difference between Y_i/n and p_i in our objective function; instead, we use the difference between their log-transformed versions, $\hat{\boldsymbol{\rho}}$ and $\boldsymbol{\rho}$. We incorporate this by weighting the optimization.

Let $w_i = w_i(Y_i, n)$ be an m by 1 vector of weights. We calculate w_i as follows:

1. We start by calculating a 95% confidence interval for each p_i using Y_i, n and Wilson confidence intervals for a Binomial proportion [82, 83]. Let the confidence interval for p_i be $[p_i^-, p_i^+]$ with centre $\tilde{p}_i$.

2. We then set $w_i = (\log(p_i^+) - \log(p_i^-))^{-1}$ provided $p_i^- \neq 0$. The width of the confidence interval after it has been transformed into the log-domain gives us an idea of how much we expect $\hat{\rho}_i$ to reasonably vary from ρ_i. Taking the reciprocals of these widths gives us the weights.

   To calculate w_i if $p_i^- = 0$, i.e. if $Y_i = 0$, we need to resort to an end correction. We calculate the confidence interval for the case where 1 measurement is successful, call this confidence interval $[p_*^-, p_*^+]$ with centre $\tilde{p}_*$. We then set

$$w_i = \frac{1}{\log(p_i^+) - \log(\tilde{p}_i)} \cdot \frac{\log(p_*^+) - \log(\tilde{p_*})}{\log(p_*^+) - \log(p_*^-)}.$$

3. We also adjust $\hat{\rho}_i$ in the optimization problem to $-\log\tilde{\rho}$, the transformed version of the centre of the Wilson confidence interval.

Applying the logarithm to the binomial confidence intervals does not produce symmetric confidence intervals, that is, $\log(p_i^+) - \log(\tilde{p}) < \log(\tilde{p}) - \log(p_i^-)$, whereas the ℓ_2 error function *is* symmetrical. Nevertheless, this approximation is better the higher the value of Y_i. For apparent transmission rates $Y_i/n > 90\%$ with 600 measurements, the

difference between the upper side of the confidence interval and the lower side is less than 3%.

Incorporating these new weights, the optimization problem to solve is now:

$$\min_{\boldsymbol{\tau}} ||\boldsymbol{\tau}||_1 + \lambda||\mathbf{w} \odot (\tilde{\boldsymbol{\rho}} - A\boldsymbol{\tau})||_2^2, \tag{6.3.3}$$

where $\odot$ denotes elementwise multiplication (the Hadamard product). After solving for τ we classify each link i as lossy if $\tau_i > 0.01$, approximately equivalent to having a loss rate of greater than 1%. We are now done; we call this method *CTDn* for *CTD with noise*. CTDn is the optimization method used to get the results in the following section.

6.4 Results

To test CTDn we start by applying it to topologies from the Internet Topology Zoo [3]. We restrict the testing to topologies with greater than 20 nodes since topologies with fewer nodes often induce trivial tomography problems. We use the following loss model: 15% of links have loss, and the transmission probability for lossy links is chosen uniformly at random between 90% and 99%. We choose 40% of the nodes in the network to be measurement nodes, and perform 600 measurements between each pair of nodes. These settings for the tests were chosen to provide challenging problems of a reasonable size without violating the sparsity assumption of relatively few links having significant loss, or having an unrealistically large number of poorly functioning links. Additionally, we show the sensitivity of the algorithm to changing these settings in the next section (Figures 6.6 – 6.14).

First, we display how the Lagrangian parameter λ affects the results of the algorithm; in particular, the number of false positives and false negatives as a proportion of the total logical links (Figures 6.2 – 6.4). The classification is into "lossy" (less than 99% transmission rate) and "good" (greater than 99% transmission rate). Correctly classifying a lossy link is considered a true positive, and correctly classifying a good (non-lossy) link is considered a true negative (since we are looking for lossy links).

Unsurprisingly, as λ increases, there are more false positives and fewer false negatives (Figures 6.2 – 6.4) because more links are diagnosed as positive. This is because the sparsity penalty function $||\boldsymbol{\tau}||_1$ is de-emphasized relative to the error part of the objective function, $||\mathbf{w} \odot (\tilde{\boldsymbol{\rho}} - A\boldsymbol{\tau})||_2^2$. Since links classified as negatives correspond to zeroes in the loss coefficients, putting a greater weight on the sparsity penalty function causes there to be more negatives in the solution to (6.3.3). The choice of measurement style seems to have little effect on the shape of the plots, however it does change the upper

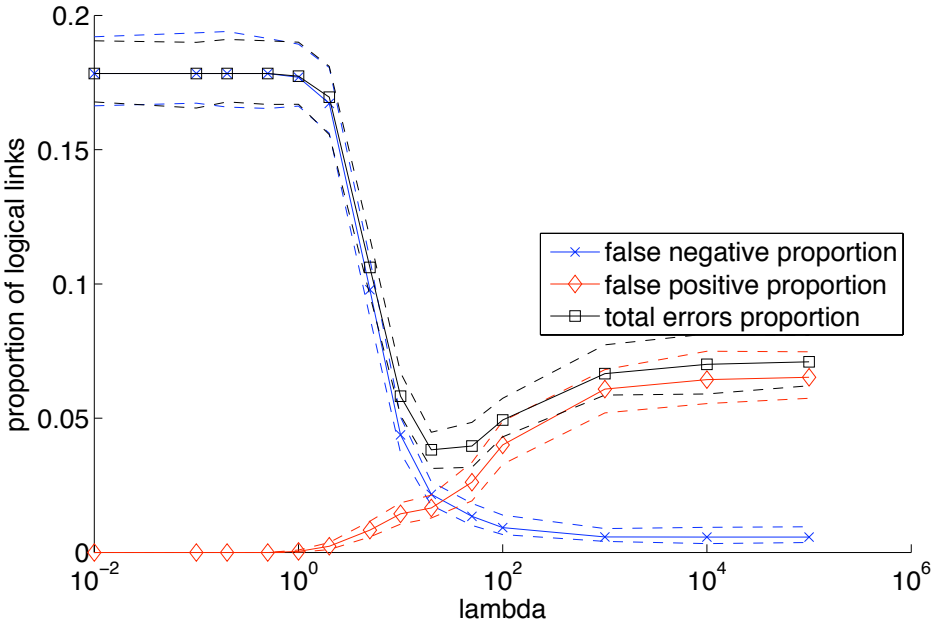


Figure 6.2: Effectiveness of CTDn for varying λ when using Measurement Style 1. Plotted are the proportions of logical links that are classified incorrectly as having no loss (*false negatives*) or loss (*false positives*). *Total* is the sum of false positives and false negatives. Note that, due to the nature of the aggregation process, on average more than 15% of the *logical links* are lossy even though only 15% of the *physical links* are lossy.

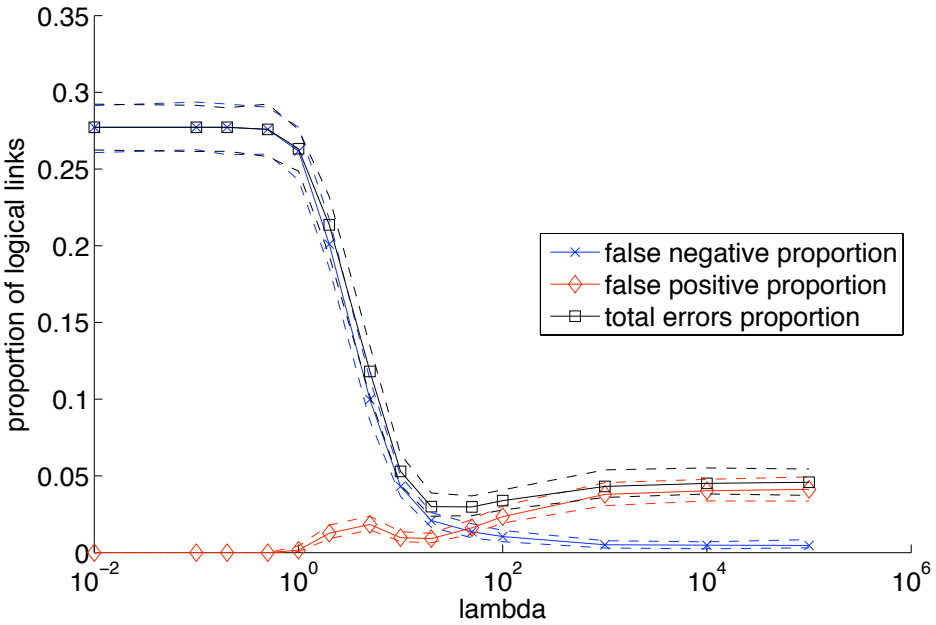


Figure 6.3: Effectiveness of CTDn for varying λ when using Measurement Style 2. Plotted are the proportions of logical links that are classified incorrectly as having no loss (*false negatives*) or loss (*false positives*).

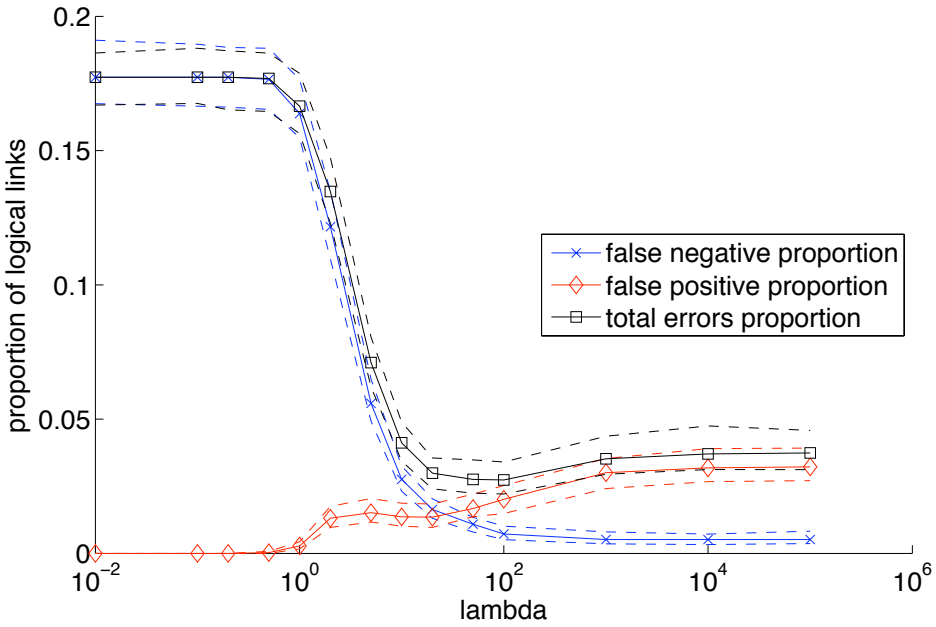


Figure 6.4: Effectiveness of CTDn for varying λ when using Measurement Style 3. Plotted are the proportions of logical links that are classified incorrectly as having no loss (*false negatives*) or loss (*false positives*).

limit for the number of false positives. This is caused by the reduced number of logical links in Measurement style 2, and consequently increased proportion of lossy logical links (for a fixed proportion of lossy physical links).

The ROC curve in Figure 6.5 summarizes the relationship between the different types of error as λ varies. Low values of λ correspond to the lower left hand corner, while high values of λ correspond to the top of the graph. As λ is increased, the emphasis on sparsity is reduced, and the emphasis on fitting the measurements (in a least-squares sense) is increased. This causes a greater number of positives due to the nature of the ℓ_2-norm. Note that the curve doesn't reach 100% true positive rate, since it only covers when λ is varied: to approach closer to 100% true positive rate the threshold on classifying values as a positive would have to be reduced as well.

Importantly, all cases give a reasonably wide range for λ, and this range is roughly consistent, making choice of an appropriate λ easy. For instance, in the sensitivity analysis in the next section we fix λ as 20.

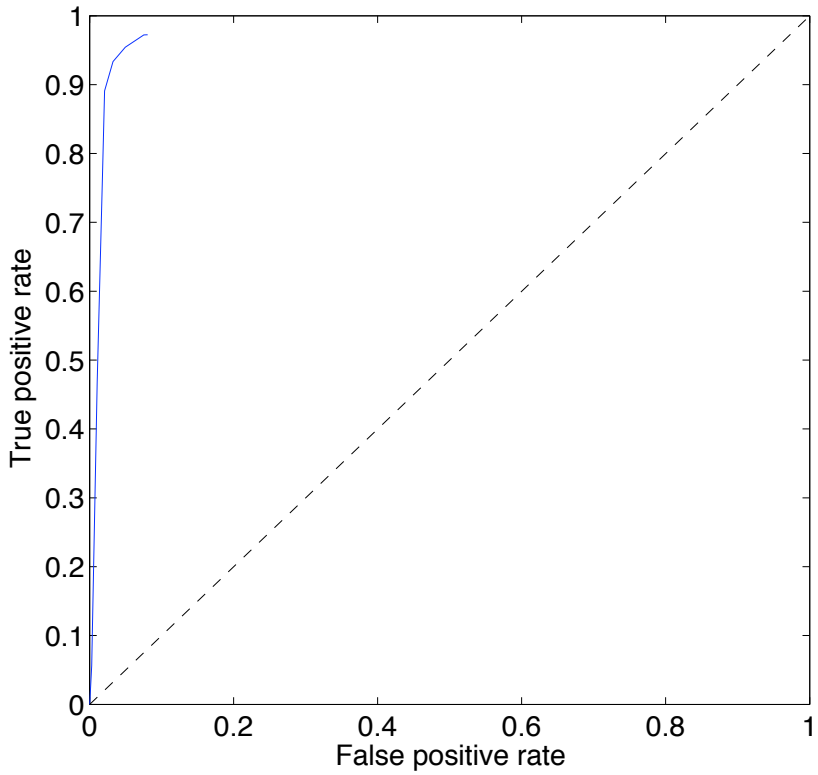


Figure 6.5: Receiver operating characteristic (ROC) curve for varying λ when using Measurement Style 1; the results for other Measurement Styles are very similar. The false-positive rate here is the number of false positives over the total number of negatives (*i.e.,* the false positives plus the true negatives). Similarly, the true-positive rate is the number of true positives divided by the number of positives. Low values of λ correspond to the lower left hand corner, while high values of λ correspond to the top of the graph.

6.5 Sensitivity analysis

In this section we show how well our classification algorithm performs when parameters of the simulations are varied. For these tests this we fix $\lambda = 20$, a tradeoff between the false-negative and false-positive rates, selected from the results in Section 6.4. For each parameter we show how the number of classification errors varies with that parameter,

for each of the 3 measurement styles. Changing the Measurement Style results in curves of similar shapes but different magnitudes.

As the number of measurements increase, the proportion of both types of classification errors decreases (Figures 6.6 – 6.8). As the number of measurements increase, errors in the measurements themselves become less relevant and the performance of CTDn tends to that of CTD. The results for large numbers of measurements (10000 measurements) is consistent with the results in Chapter 5. One large source of errors is that CTDn may be finding a sparser solution (one with fewer lossy links) consistent with the same path transmission probabilities. This is a source of error that is impossible to avoid, and it gets even worse with low numbers of measurements, since the measurement variation allows for a wider range of $\boldsymbol{\tau}$ to be consistent with the measurements.

An increase in the proportion of lossy links (positives) predictably causes an overall loss in classification accuracy (Figures 6.9 – 6.11). As the proportion of lossy links gets higher, the sparsity assumption becomes less valid, and becomes a less effective tool for diagnosing the correct classification of lossy links. For proportions of lossy links less than about 0.4, there's a roughly linear increase in the number of errors. Naturally, the higher the number of lossy links (considered as positives), the less likely a positive will be a false positive. It is not at all surprising that high numbers of lossy links causes high numbers of false negatives, since the algorithm is trying to find an explanation for the measurements with as few lossy links as possible, so there will be many fewer links classified as positives (lossy) than there actually are. However, with Measurement Style 2 and Measurement Style 3 (Figures 6.10 and 6.11), once the proportion of lossy links becomes very high the number of errors actually starts decreasing. This is presumably because the sparsity component of the optimisation is largely overcome once there are very few "good" links, and CTDn starts to guess that nearly all the links are lossy.

Note that in Figures 6.9 – 6.11 the lossy links are measured as a proportion of the physical links (rather than as a proportion of logical links) since that is the actual input to the simulation. Different logical link aggregations can result in a different number of lossy logical links for a fixed number of lossy physical links. The errors are measured as a proportion of the logical links for consistency with all the other results.

We see in Figures 6.12 – 6.14 the effect of minimum transmission probability for links on the number of errors made by CTDn. The maximum transmission probability is fixed at 99% and transmission probabilities for lossy links are generated uniformly at random in this interval. Interestingly, the minimum transmission probability seems to have little affect on the numbers of false negatives (good links described as being lossy), except when it is very close to the maximum transmission probability. The false positives are much higher with lower minimum transmission probabilities. This is likely due to the effect described previously where the high variance of transmission measurements on lossy links can appear equivalent to low rates of loss on adjacent links.

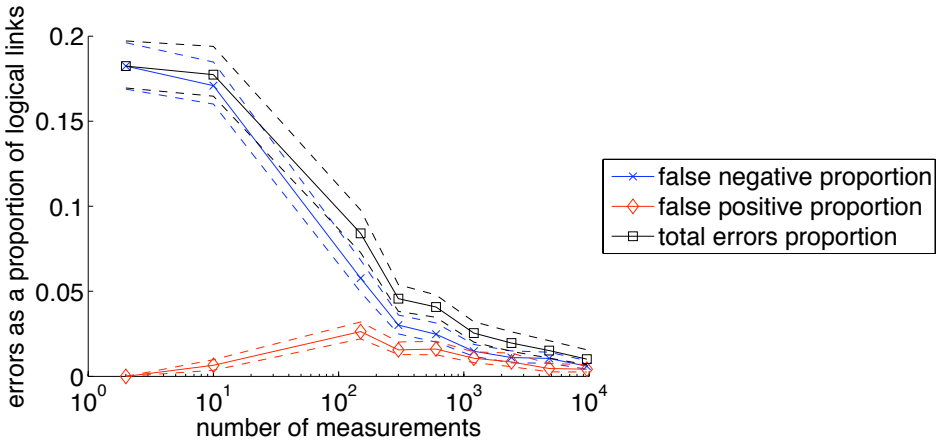


Figure 6.6: Plot of errors in classification versus the number of measurements for Measurement Style 1 (on a log scale). The number of measurements used for the other simulations in this chapter is 600. As the number of measurements increases, the relative measurement error decreases and the performance of CTDn tends to that of CTD.

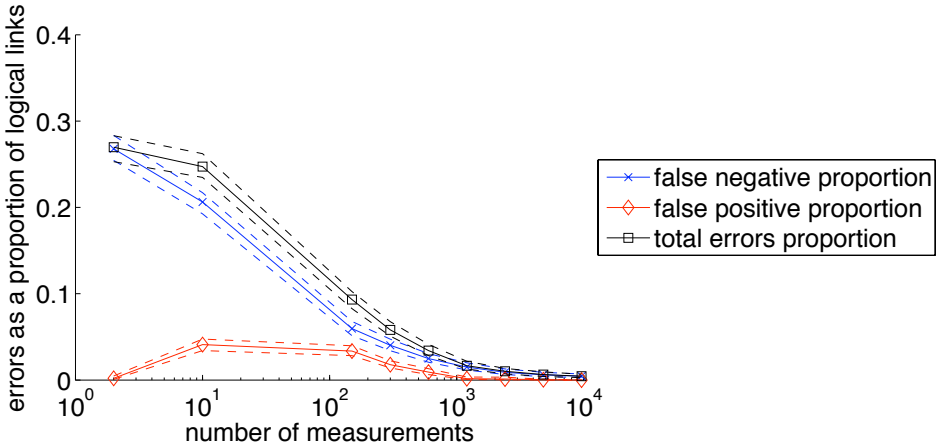


Figure 6.7: Plot of errors in classification versus the number of measurements for Measurement Style 2. The number of measurements used for the other simulations in this chapter is 600. Similar to Figure 6.6 except that in this case the errors tend to 0 as the number of measurements increases, consistent with the results in Chapter 5.

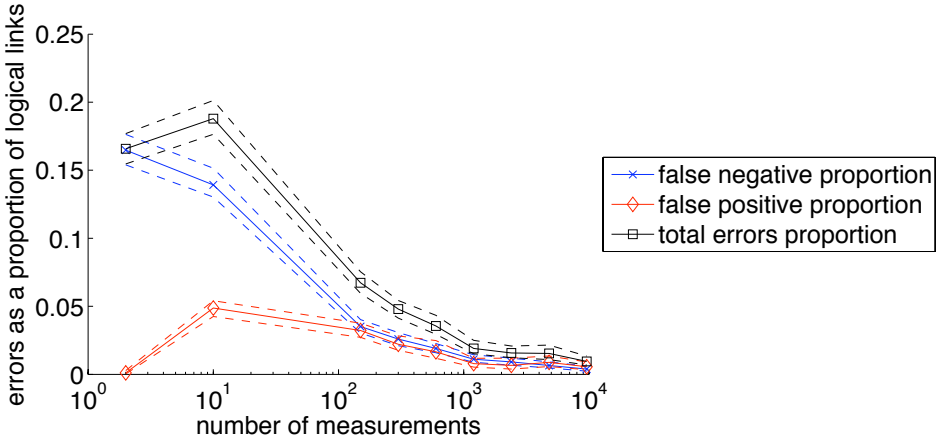


Figure 6.8: Plot of errors in classification versus the number of measurements for Measurement Style 3. The number of measurements used for the other simulations in this chapter is 600.

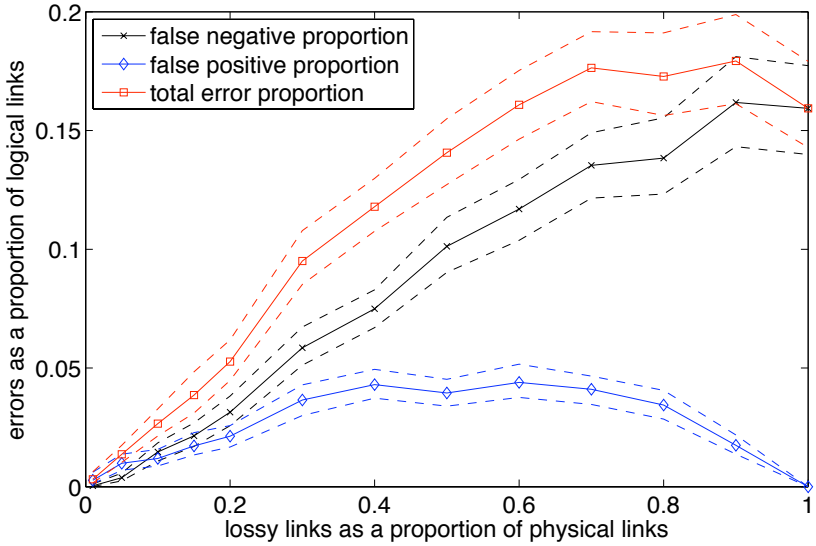


Figure 6.9: Plot of errors in classification versus the proportion of lossy links for Measurement Style 1. For comparison, in the other simulations presented here the proportion of lossy links is fixed at 0.15.

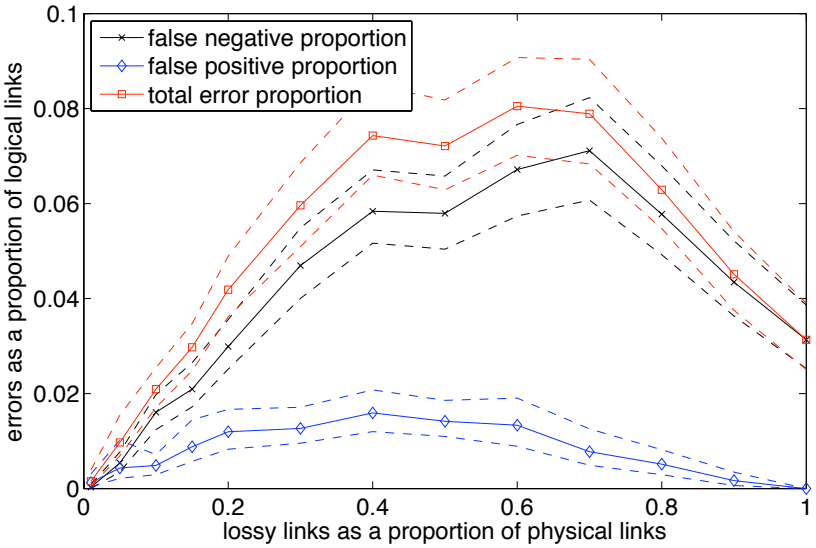


Figure 6.10: Plot of errors in classification versus the proportion of lossy links for Measurement Style 2.

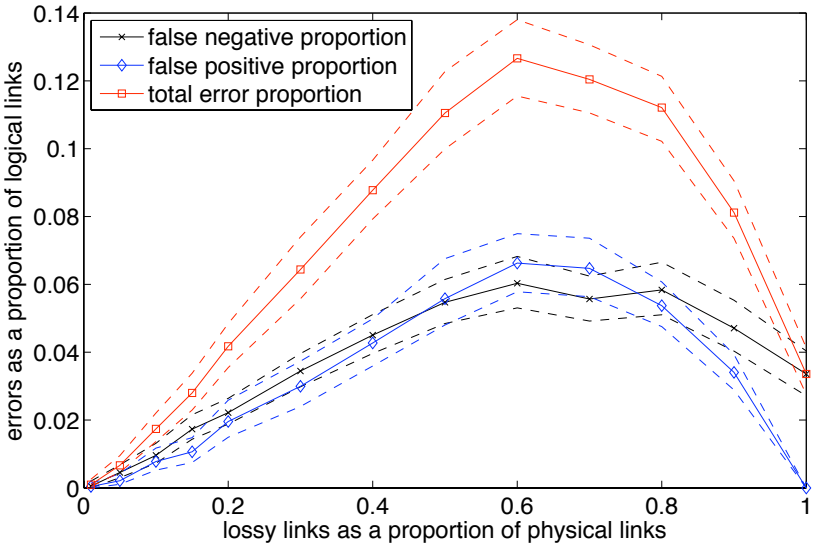


Figure 6.11: Plot of errors in classification versus the proportion of lossy links for Measurement Style 3.

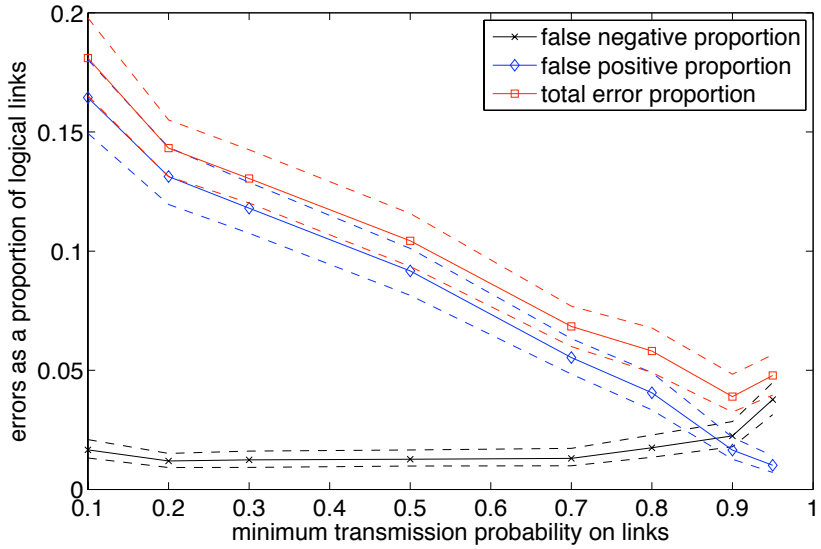


Figure 6.12: Plot of errors in classification versus the minimum transmission probability for (lossy) links for Measurement Style 1. The maximum transmission probability is fixed at 99%.

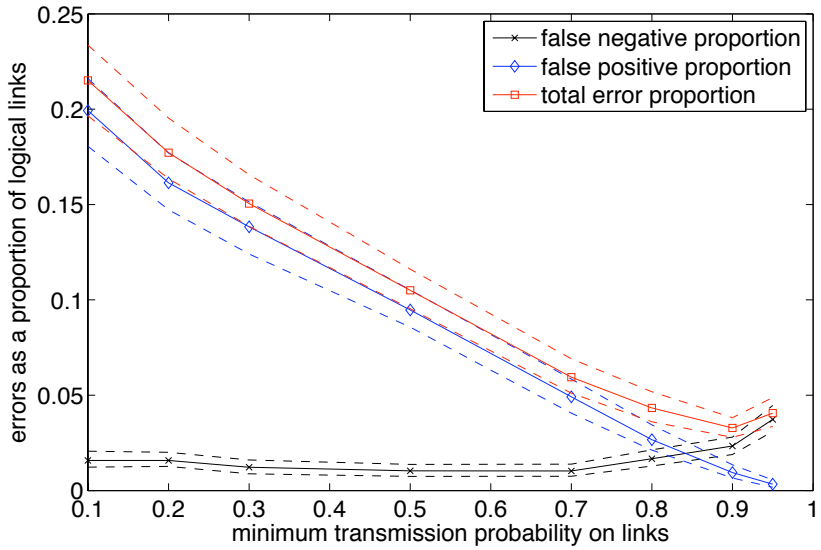


Figure 6.13: Plot of errors in classification versus the minimum transmission probability for (lossy) links for Measurement Style 2. Measurement Style 2 has a slightly higher proportion of logical links as errors (than Measurement Styles 1 or 3), likely due to having a lower number of logical links.

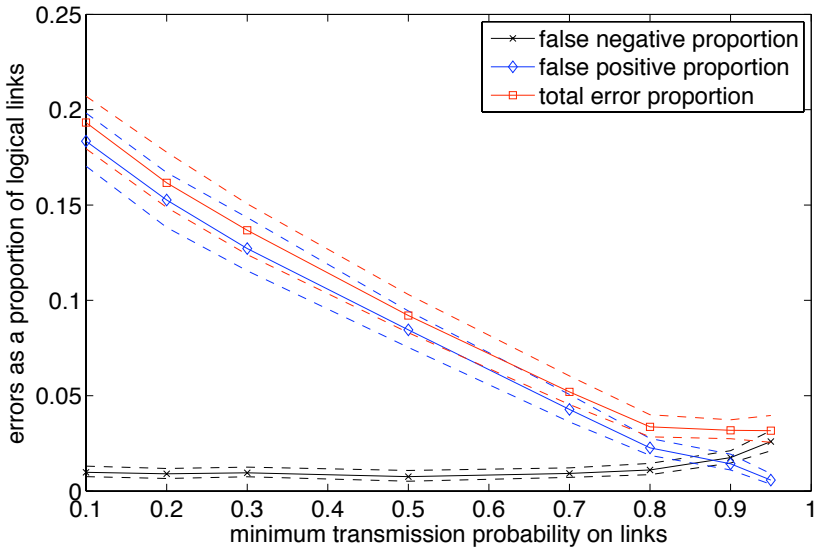


Figure 6.14: Plot of errors in classification versus the minimum transmission probability for (lossy) links for Measurement Style 3.

6.6 Results on synthetic topologies

In this section we present the results when CTDn is applied to synthetic topologies generated with COLD. Results are presented for each Measurement Style (1–3) and each Scenario (1–3), as in Chapter 5. In this case, the classification errors are plotted versus the number of measurement nodes, to be consistent with Chapter 5.

In each case (Figures 6.15 – 6.23) the average number of errors is small, but such a low average number of classification errors causes a higher relative variance, and thus apparently wider 95% confidence intervals for the mean. Modulo these uncertainties, the relative proportion of logical links that are misclassified generally decreases with increasing number of measurement nodes. Increasing number of measurement nodes typically means an increased number of logical links, so the decreasing proportion of errors indicates better information about each logical link, much as in the noiseless case (Chapters 4 and 5).

Despite the addition of noise, the average number of misclassified links remains very low, under 5% in most cases. While it is greater than in the noiseless case, this still represents a successful adaption to the noisy tomography problem.

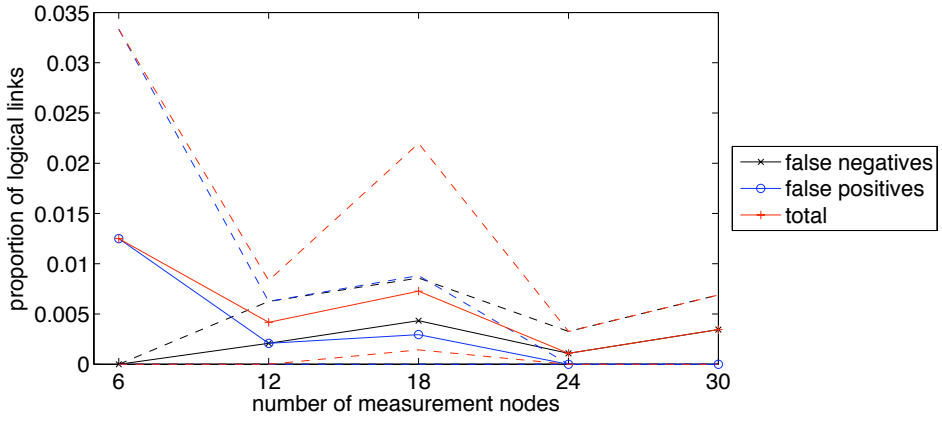


Figure 6.15: Error proportion versus number of measurement sources. Scenario 1, Measurement Style 1.

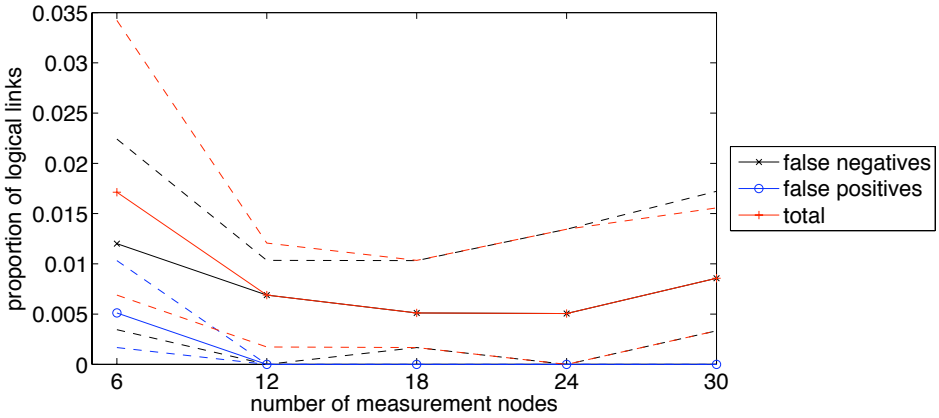


Figure 6.16: Error proportion versus number of measurement sources. Scenario 1, Measurement Style 2.

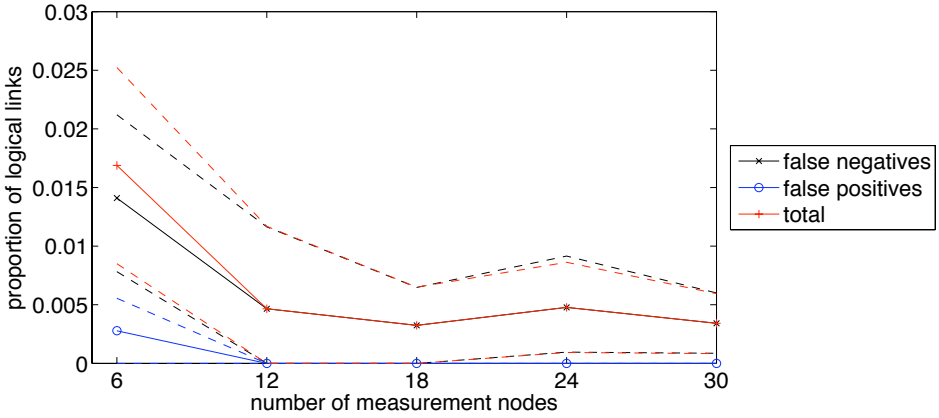


Figure 6.17: Error proportion versus number of measurement sources. Scenario 1, Measurement Style 3.

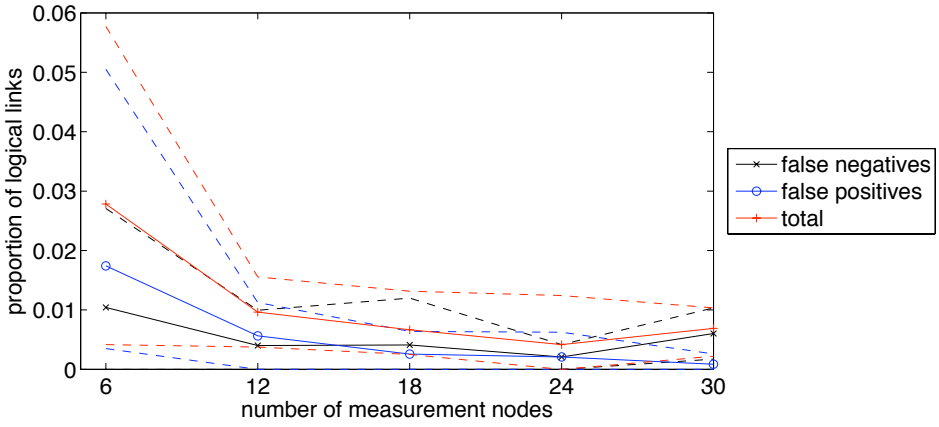


Figure 6.18: Error proportion versus number of measurement sources. Scenario 2, Measurement Style 1.

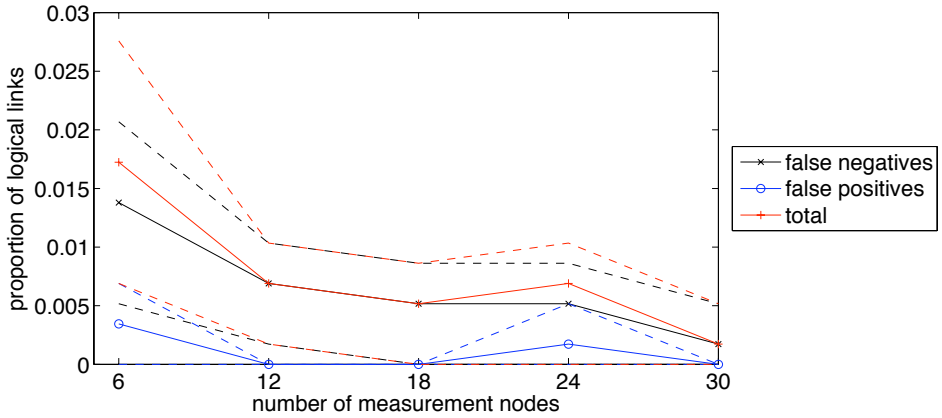


Figure 6.19: Error proportion versus number of measurement sources. Scenario 2, Measurement Style 2.

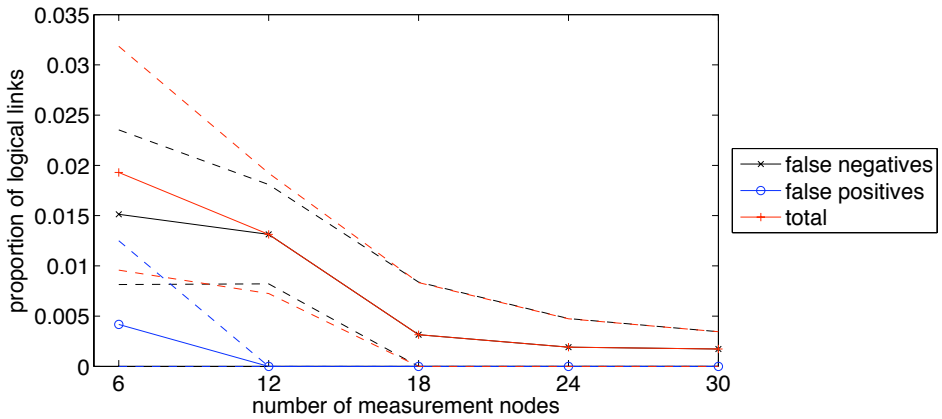


Figure 6.20: Error proportion versus number of measurement sources. Scenario 2, Measurement Style 3.

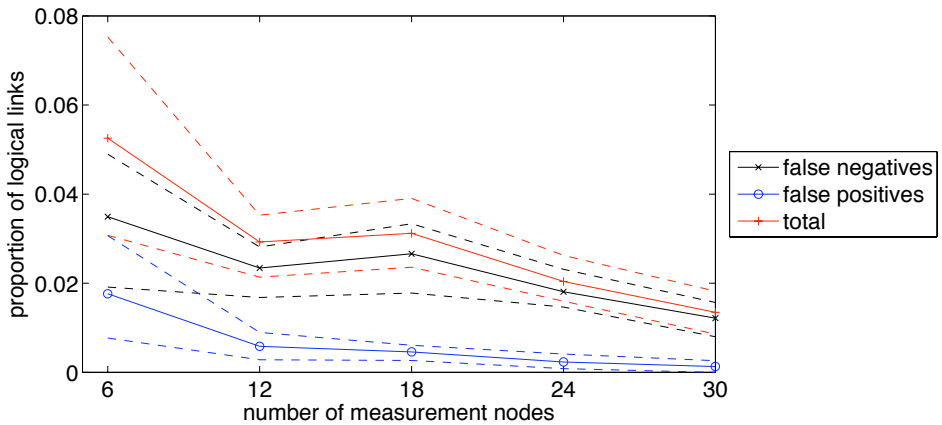


Figure 6.21: Error proportion versus number of measurement sources. Scenario 3, Measurement Style 1.

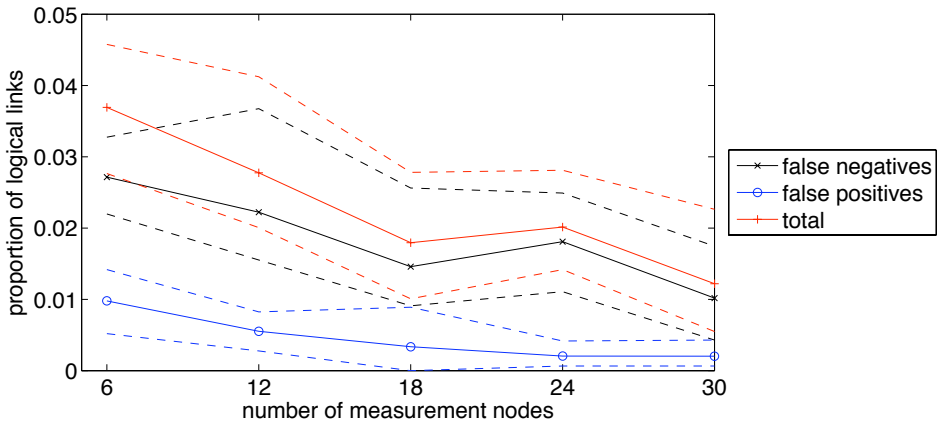


Figure 6.22: Error proportion versus number of measurement sources. Scenario 3, Measurement Style 2.

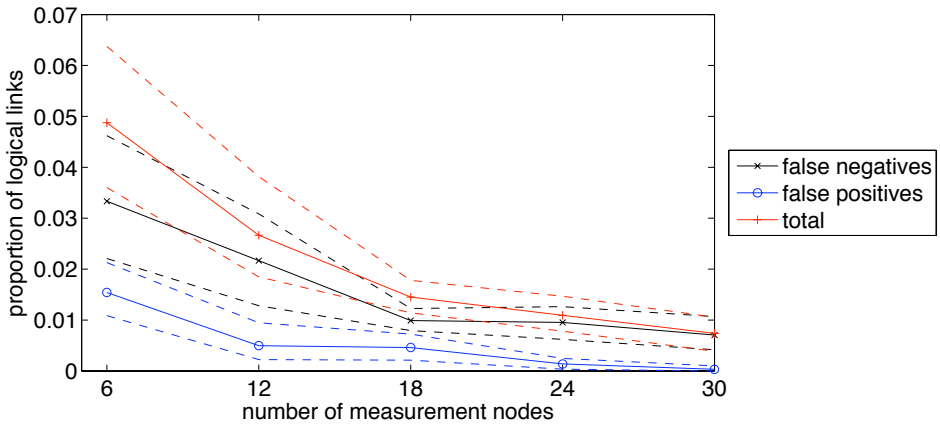


Figure 6.23: Error proportion versus number of measurement sources. Scenario 3, Measurement Style 3.

6.7 Conclusion

Without a means to deal with measurement noise, it can be difficult to apply a tomography algorithm in practice. Even with one measurement per second for ten minutes, this only gives 600 measurements, not enough to estimate loss rates to very high degrees of accuracy. To address the problem of measurement noise we propose an algorithm, CTDn, inspired by a common optimization technique from compressive sensing or feature selection but modified to suit the link-loss tomography problem. Link loss tomography has a variety of challenges: incorporating information from a large number of paths; links with high loss probability can mask links with lower loss probability due to their high variance; even estimating loss on individual links is not trivial. The algorithm presented here performs well across a wide range of simulation scenarios, especially considering the wide variety of sources of error.

Chapter 7

Conclusion

This thesis addresses the problem of link loss tomography on general networks. The problem is difficult for several reasons, including the need to merge information from several sources of measurement, the potentially large number of links and measured paths, and the difficulty of making accurate loss rate estimates from binary measurements of success or failure. Further, the problem is frequently undetermined. To deal with these challenges, we use the observation that links with high rates of loss are relatively rare in a functional wired network, a form of sparsity. This allows us to employ a relatively new field of mathematics, compressive sensing, a field that deals with the recovery of information when the measurement made are incomplete or compressed. We present two algorithms utilizing compressive sensing: CTD and CTDn.

Since the success or failure of these tomographic methods depends on the underlying network topology, we must provide a set of topologies on which to test CTD and CTDn. Unfortunately, real-life topologies provided by measurement studies are incomplete or inaccurate in many cases since the underlying protocols and tools used in measurement are rarely designed for topology discovery [46]. Instead, we use maps provided by network operators, collected and digitized in the Topology Zoo [3]. While this provides a good source of topologies, there are still insufficient numbers to allow a rigorous statistical testing of our tomography methods.

To provide a larger number of network topologies we turn to our own method of topology synthesis: COLD. COLD provides realistic PoP-level topology designs by optimizing constraints modelled on real-world economic constraints. These designs are then used to build router-level network topologies by templated design using graph products. COLD produces realistic, context-rich network topologies that can be tuned to reflect adaptation to a wide variety of real-world situations in which networks are built and evolve.

We then present the topology algorithms CTD and CTDn. CTD works under the assumption that we have access to very accurate loss rate data along paths. It utilizes aspects of the structure of routing matrices to perform fast size reductions of the tomography problem prior to using compressive sensing methods. The reductions dramatically increase the speed of the overall algorithm, and provide information about which components of the solution vector are known, and which are best-effort estimates. CTD solves the link tomography problem very successfully, and is limited more by fundamental limitations in the measurements than by its own shortcomings.

Unlike CTD, CTDn is improved to use a limited number of measurements, and consequently to deal with noisy estimates of loss rates. The errors in path measurement can be magnified when attempting to find link loss rates, and links with high rates of loss can obscure loss on link with lower rates of loss. Despite these difficulties, CTDn classifies link loss rates successfully over 95% of the time in most of the measured scenarios.

Future work could involve modelling physical population distributions and geographic features (*e.g.,* coastlines and mountain ranges) that affect both link costs and human populations (and hence PoP locations and populations) to provide a more realistic context for COLD.

Other future work could include increasing the efficiency of CTDn by integrating more of the pre-processing features of CTD. Other work could focus on the router-level topologies created from the PoP-level topologies using graph products. We could use characteristics (like traffic throughput) of links (and PoPs) in the PoP-level networks to determine the graph products (and graphs) used when generating the router-level networks. The resulting router-level topologies could be compared to those in the Internet Topology Zoo. Also, we could investigate the effects of different choices of graph-product on how CTD performs on router-level topologies generated by COLD.

Bibliography

[1] R. Bowden, "Cold: Pop-level topology synthesis," in *ACM International Conference on emerging Network EXperiments and Technologies (CoNEXT)*, 2014.

[2] E. Parsonage, H. X. Nguyen, R. Bowden, S. Knight, N. Falkner, and M. Roughan, "Generalized graph products for network design and analysis," in *19th IEEE International Conference on Network Protocols*, 2011.

[3] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," in *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9. IEEE, October 2011, pp. 1765–1775.

[4] R. Bowden, H. X. Nguyen, N. Falkner, S. Knight, and M. Roughan, "Planarity of data networks," in *23rd International Teletraffic Congress (ITC)*. IEEE, 2011, pp. 254–261.

[5] J. Sommers, R. A. Bowden, B. Eriksson, P. Barford, M. Roughan, and N. G. Duffield, "Efficient network-wide flow record generation," in *IEEE Conference on Computer Communications (INFOCOM)*, 2011, pp. 2363–2371.

[6] B. Eriksson, P. Barford, R. Bowden, N. Duffield, J. Sommers, and M. Roughan, "Basisdetect: a model-based network event detection framework," in *ACM SIGCOMM Conference on Internet measurement (IMC)*. ACM, 2010, pp. 451–464.

[7] Y. Vardi, "Network Tomography: Estimating Source-Destination Traffic Intensities from Link Data." *Journal of the American Statistical Association (0162-1459)*, vol. 91, no. 433, 1996.

[8] B. Eriksson, G. Dasarathy, P. Barford, and R. Nowak, "Toward the practical use of network tomography for internet topology discovery," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2010, pp. 1–9.

[9] D. Ghita, H. Nguyen, M. Kurant, K. Argyraki, and P. Thiran, "Netscope: practical network loss tomography," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2010, pp. 1–9.

[10] M. Coates, R. Hero, A. Nowak, and B. Yu, "Internet tomography," *IEEE Signal Processing Magazine*, vol. 19, no. 3, pp. 47–65, May 2002.

[11] N. Duffield, F. LoPresti, V. Paxson, and D. Towsley, "Network loss tomography using striped unicast probes," *IEEE/ACM Transactions on Networking*, vol. 14, no. 46, pp. 697–710, Aug. 2000.

[12] Y. Shavitt, X. Sun, A. Wool, and B. Yener, "Computing the unmeasured: An algebraic approach to Internet mapping," in *IEEE Conference on Computer Communications (INFOCOM)*, 2001.

[13] M. Coates and D. Nowak, "Network tomography for internal delay estimation," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, May 2001.

[14] N. Duffield, "Network tomography of binary network performance characteristics," *IEEE Transactions on Information Theory*, vol. 52, no. 12, pp. 5373–5388, December 2006.

[15] V. Arya and D. Veitch, "Sparsity without the complexity: Loss localisation using tree measurements," in *NETWORKING 2012*. Springer, 2012, pp. 289–303.

[16] M. Rabbat, R. Nowak, and M. Coates, "Multiple source, multiple destination network tomography," in *IEEE Conference on Computer Communications (INFOCOM)*, 2004.

[17] A. Adams, T. Bu, T. Friedman, J. Horowitz, D. Towsley, R. Cáceres, N. Duffield, F. Presti, S. Moon, and V. Paxson, "The use of end-to-end multicast measurements for characterizing internal network behavior," *Communications Magazine, IEEE*, vol. 38, no. 5, pp. 152–159, 2000.

[18] R. Cáceres, N. Duffield, J. Horowitz, and D. Towsley, "Multicast-based inference of network-internal loss characteristics," *IEEE Transactions on Information Theory*, vol. 45, no. 7, pp. 2462–2480, 1999.

[19] T. Bu, N. Duffield, F. Presti, and D. Towsley, "Network tomography on general topologies," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 1. ACM, 2002, pp. 21–30.

[20] M. Coates, R. Nowak *et al.*, "Network loss inference using unicast end-to-end measurement," in *Proc. ITC Conf. IP Traffic, Modeling and Management*, 2000, pp. 28–1.

[21] H. Nguyen and P. Thiran, "Network loss inference with second order statistics of end-to-end flows," in *ACM SIGCOMM Conference on Internet measurement (IMC)*. ACM, 2007, pp. 227–240.

[22] D. Ghita, K. Argyraki, and P. Thiran, "Network tomography on correlated links," in *ACM SIGCOMM Conference on Internet measurement (IMC)*, November 1-3 2010, pp. 225–238.

[23] D. Donoho, "For most large underdetermined systems of linear equations the minimal ℓ1-norm solution is also the sparsest solution," *Comm. Pure Appl. Math*, vol. 59, pp. 797–829, 2004.

[24] R. Berinde, A. C. Gilbert, P. Indyk, H. Karloff, and M. J. Strauss, "Combining geometry and combinatorics: A unified approach to sparse signal recovery," in *46th Annual Allerton Conference on Communication, Control, and Computing*, Sep. 2008, pp. 798 – 805.

[25] E. J. Candès and T. Tao., "Near-optimal signal recovery from random projections and universal encoding strategies," *IEEE Transactions on Information Theory*, vol. 52, no. 12, pp. 5406–5425, 2004.

[26] Y. Tsaig and D. Donoho, "Extensions of compressed sensing," *Signal Processing*, vol. 86, no. 3, pp. 549–571, March 2006.

[27] J. Tropp, "Greed is good: algorithmic results for sparse approximation," *IEEE Transactions on Information Theory*, vol. 50, no. 10, pp. 2231–2242, October 2004.

[28] R. Baraniuk, M. Davenport, R. Devore, and M. Wakin, "A simple proof of the restricted isometry property for random matrices," *Constructive Approximation*, 2008.

[29] V. Chandar, "A negative result concerning explicit matrices with the restricted isometry property," *preprint*, 2008.

[30] S. Knight, H. Nguyen, N. Falkner, R. Bowden, M. Roughan, and E. Parsonage, "The Internet topology zoo." [Online]. Available: www.topology-zoo.org

[31] H. Ringberg, M. Roughan, and J. Rexford, "The need for simulation in evaluating anomaly detectors," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 1, pp. 55–59, January 2008.

[32] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *J. Mach. Learn. Res.*, vol. 11, pp. 985–1042, March 2010.

[33] P. Erdös and A. Rényi, "On the evolution of random graphs," *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, vol. 5, pp. 17–61, 1960.

[34] B. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, December 1988.

[35] E. W. Zegura, K. L. Calvert, and M. J. Donahoo, "A quantitative comparison of graph-based models for Internet topology," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 770–783, December 1997.

[36] C. Jin, Q. Chen, and S. Jamin, "Inet: Internet Topology Generator," University of Michigan at Ann Arbor, Tech. Rep. CSE-TR-433-00, 2000.

[37] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: an approach to universal topology generation," in *Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2001, pp. 346–353.

[38] F. Chung and L. Lu, "The average distance in a random graph with given expected degrees," *Internet Mathematics*, vol. 1, no. 1, pp. 91–113, 2004.

[39] W. Aiello, F. Chung, and L. Lu, "A random graph model for massive graphs," in *ACM Symposium on Theory of computing*, 2000, pp. 171–180.

[40] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet map discovery," in *IEEE Conference on Computer Communications (INFOCOM)*, 2000, pp. 1371–1380.

[41] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos, "Power laws and the AS-level Internet topology," *IEEE/ACM Transactions on Networking*, vol. 11, pp. 514–524, August 2003.

[42] "Skitter, Cooperative Association for Internet Data Access." [Online]. Available: www.caida.org/tools/measurement/skitter/

[43] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, pp. 133–145, 2002.

[44] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the Internet topology," in *ACM SIGCOMM*, New York, NY, USA, 1999, pp. 251–262.

[45] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz, "Characterizing the Internet hierarchy from multiple vantage points," in *IEEE Conference on Computer Communications (INFOCOM)*, 2002, pp. 618–627.

[46] W. Willinger, D. Alderson, and J. Doyle, "Mathematics and the Internet: A source of enormous confusion and great potential." *Notices of the AMS*, vol. 56, no. 5, pp. 586–599, 2009.

[47] W. Willinger, R. Govindan, S. Jamin, V. Paxson, and S. Shenker, "Scaling phenomena in the Internet: Critically examining criticality," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. Suppl 1, pp. 2573–2580, 2002.

[48] W. Willinger, D. Alderson, J. C. Doyle, and L. Li, "More "normal" than normal: scaling distributions and complex systems," in *Proceedings of the 36th conference on Winter simulation*, ser. WSC '04. Winter Simulation Conference, 2004, pp. 130–141.

[49] W. Willinger, D. Alderson, and L. Li, "A pragmatic approach to dealing with high-variability in network measurements," in *ACM SIGCOMM IMC*, 2004, pp. 88–100.

[50] J. Doyle, D. Anderson, L. Li, S. Low, M. Roughan, S. Shalunov, R. Tanaka, and W. Willinger, "The "robust yet fragile" nature of the Internet," in *Proceedings of the National Academy of Sciences of the United States of America*, 2006.

[51] L. Li, D. Alderson, W. Willinger, and J. Doyle, "A first-principles approach to understanding the Internet's router-level topology," in *ACM SIGCOMM*, New York, NY, USA, 2004, pp. 3–14.

[52] A. Fabrikant, E. Koutsoupias, and C. Papadimitriou, "Heuristically optimized trade-offs: A new paradigm for power laws in the Internet," in *Automata, Languages and Programming*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2002, vol. 2380, pp. 781–781.

[53] D. Alderson, L. Li, W. Willinger, and J. Doyle, "Understanding Internet topology: principles, models, and validation," *IEEE/ACM Transactions on Networking*, vol. 13, pp. 1205–1218, 2005.

[54] Cisco, "ISP network design," 2005. [Online]. Available: ws.edu.isoc.org/data/2005/41363436042fc2b563533b/d1-6up.pdf

[55] V. Gill, "Analysis of design decisions in a 10G backbone." [Online]. Available: www.nanog.org/meetings/nanog34/presentations/gill.pdf

[56] M. Morris, "Network design templates," www.networkworld.com/community/blog/network-design-templates, July 18 2007.

[57] E. Parsonage, H. X. Nguyen, R. Bowden, S. Knight, N. J. Falkner, and M. Roughan, "Generalized graph products for network design and analysis," in *Proc. of the 19th IEEE ICNP*, Vancouver, CA, October 2011.

[58] Y. Shavitt and N. Zilberman, "A structural approach for PoP geo-location," in *IEEE Conference on Computer Communications (INFOCOM)*, 2010.

[59] W. Aiello, F. Chung, and L. Lu, "A random graph model for power law graphs," *Experimental Mathematics*, vol. 10, no. 1, pp. 53–66, 2001.

[60] P. Mahadevan, D. Krioukov, K. Fall, and A. Vahdat, "Systematic topology analysis and generation using degree correlations," in *ACM SIGCOMM*, 2006, pp. 135–146.

[61] P. Mahadevan, C. Hubble, D. Krioukov, B. Huffaker, and A. Vahdat, "Orbis: rescaling degree correlations to generate annotated Internet topologies," in *ACM SIGCOMM*, 2007, pp. 325–336.

[62] P. Pöyhönen, "A tentative model for the volume of trade between countries," *Weltwirtschaftliches Archive*, vol. 90, pp. 93–100, 1963.

[63] J. Kowalski and B. Warfield, "Modeling traffic demand between nodes in a telecommunications network," in *ATNAC'95*, 1995.

[64] Y. Zhang, M. Roughan, C. Lund, and D. Donoho, "An information-theoretic approach to traffic matrix estimation," in *ACM SIGCOMM*, Karlsruhe, Germany, August 2003, pp. 301–312.

[65] D. Alderson, H. Chang, M. Roughan, S. Uhlig, and W. Willinger, "The many facets of Internet topology and traffic," *Networks and Heterogeneous Media*, vol. 1, no. 4, pp. 569–600, December 2006.

[66] K. N. Oikonomou, "Analytic forms for most likely matrices derived from incomplete information," *Int. J. Systems Science*, vol. 43, no. 3, pp. 443–458, Sep. 2010.

[67] T. Benson, A. Akella, and D. Maltz, "Unraveling the complexity of network management," *Proc. NSDI*, 2009.

[68] R. Bowden. (2012) COLD: A method for synthesising data network topologies. [Online]. Available: http://github.com/rhysbowden/COLD/

[69] K. Yoshida, Y. Kikuchi, M. Yamamoto, Y. Fujii, K. Nagami, I. Nakagawa, and H. Esaki, "Inferring PoP-level ISP topology through end-to-end delay measurement," in *PAM 2009*, 2009, pp. 35–44.

[70] V. Padmanabhan, L. Qiu, and H. Wang, "Server-based inference of internet link lossiness," in *IEEE Conference on Computer Communications (INFOCOM)*, vol. 1. IEEE, 2003, pp. 145–155.

[71] D. Takhar, J. N. Laska, M. B. Wakin, M. F. Duarte, D. Baron, S. Sarvotham, K. F. Kelly, and R. G. Baraniuk, "A new compressive imaging camera architecture using optical-domain compression," in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, vol. 6065, Feb. 2006, pp. 43–52.

[72] F. K. Hwang, "A method for detecting all defective members in a population by group testing," *Journal of the American Statistical Association*, vol. 67, no. 339, pp. 605–608, 1972.

[73] F. Parvaresh, H. Vikalo, S. Misra, and B. Hassibi, "Recovering sparse signals using sparse measurement matrices in compressed DNA microarrays," *IEEE Journal of Selected Topics in Signal Processing*, vol. 2, no. 3, pp. 275–285, June 2008.

[74] W. Xu, E. Mallada, and A. Tang, "Compressive sensing over graphs," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2011, pp. 2087–2095.

[75] H. Mahyar, H. R. Rabiee, and Z. S. Hashemifar, "Ucs-nt: An unbiased compressive sensing framework for network tomography," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013, pp. 4534–4538.

[76] M. Firooz and S. Roy, "Link delay estimation via expander graphs," *IEEE Transactions on Communications*, 2014.

[77] W. Wallis, *A beginner's guide to graph theory*, 2nd ed. Birkhäuser, 2007.

[78] M. Rudelson and R. Vershynin, "Geometric approach to error-correcting codes and reconstruction of signals," *Int Math Res Notices*, vol. 2005, no. 64, pp. 4019–4041, 2005. [Online]. Available: imrn.oxfordjournals.org/cgi/content/abstract/2005/64/4019

[79] Y. Pati, R. Rezaiifar, and P. Krishnaprasad, "Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition," in *Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers*, vol. 1, November 1993, pp. 40–44.

[80] E. J. Candès and J. Romberg and T. Tao., "Stable signal recovery from incomplete and inaccurate measurements," *Comm. Pure Appl. Math.*, vol. 59, pp. 1207–1223, 2005.

[81] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2001.

[82] A. Agresti and B. A. Coull, "Approximate is better than "exact" for interval estimation of binomial proportions," *The American Statistician*, vol. 52, no. 2, pp. 119–126, 1998. [Online]. Available: www.jstor.org/stable/2685469

[83] L. D. Brown, T. T. Cai, and A. DasGupta, "Interval estimation for a binomial proportion," *Statistical Science*, vol. 16, no. 2, pp. 101–133, 2001.

[84] D. L. Donoho, M. Elad, and V. N. Temlyakov, "Stable recovery of sparse overcomplete representations in the presence of noise," *IEEE Transactions on Information Theory*, vol. 52, no. 1, pp. 6–18, 2006.

[85] K. Koh, S. Kim, and S. Boyd, "An interior-point method for large-scale l1-regularized logistic regression," *Journal of Machine learning research*, vol. 8, no. 8, pp. 1519–1555, 2007.

[86] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM journal on scientific computing*, vol. 20, no. 1, pp. 33–61, 1998.

[87] J. A. Tropp, "Just relax: Convex programming methods for identifying sparse signals in noise," *Information Theory, IEEE Transactions on*, vol. 52, no. 3, pp. 1030–1051, 2006.

[88] E. J. Candes, J. K. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Communications on pure and applied mathematics*, vol. 59, no. 8, pp. 1207–1223, 2006.

[89] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[90] J. Lv and Y. Fan, "A unified approach to model selection and sparse recovery using regularized least squares," *The Annals of Statistics*, vol. 37, no. 6A, pp. 3498–3528, 2009.

[91] T. Matsuda, M. Nagahara, and K. Hayashi, "Link quality classifier with compressed sensing based on ell_1-ell_2 optimization," *IEEE Communications Letters*, vol. 15, no. 10, pp. 1117–1119, 2011.