COLD: PoP-level Network Topology Synthesis

Rhys Bowden
University of Adelaide
Adelaide, Australia
rhysbowden@gmail.com

Matthew Roughan
University of Adelaide
Adelaide, Australia
matthew.roughan@
adelaide.edu.au

Nigel Bean
University of Adelaide
Adelaide, Australia
nigel.bean@
adelaide.edu.au

ABSTRACT

Network topology synthesis seeks methods to generate large numbers of example network topologies primarily for use in simulation. It is a topic that has received much attention over the years, underlying which is a conflict between randomness and design. Random graphs are appealing because they are simple and avoid the messy details that plague real networks. However real networks *are* messy, because network operators design their networks in the context of complex technological constraints, costs, and goals. When random models have been used they often produce patently unrealistic networks that only match a few artificial connectivity statistics of real networks: the features that make the network useful and interesting are ignored. At best a network divorced from context is a purely mathematical object with no meaning or utility. At worst it can be completely misleading. However, design alone cannot generate an ensemble of networks with the variability needed in simulation. We need to balance design and randomness in a way that generates reasonable networks with given characteristics and predictable variability. This paper presents such a method, Combined Optimization and Layered Design (COLD), incorporating randomness and design principles to create ensembles of PoP-level synthetic networks.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: Network Architecture and Design—*Network Topology*; C.2.5 [**Computer Communication Networks**]: Local and Wide-Area Networks

General Terms

Design,Theory

Keywords

Topology generation, Heuristically Optimal Topologies

CoNEXT'14, December 2–5, 2014, Sydney, Australia.

http://dx.doi.org/10.1145/2674005.2675012.

1. INTRODUCTION

> Everything should be made as simple as it can be — but not simpler.
>
> – *Attributed to A. Einstein*

The core problem of network synthesis is: *take one or more network topologies, and produce a larger set of topologies that is similar in some fundamental way.* 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.

The problem imposes requirements, with most attention historically aimed at "similarity", because it is difficult to universally define for all types of network. 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 paper is concerned with data-communications networks.

Our approach is motivated by Li *et al.* [1] 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 HOT router-level synthesis described in [1] to multi-layered networks, where it takes a different form. Large data networks are not designed by a pure router-level optimization, they use hierarchy and templated design to simplify the design process [2–4]. The most common form of hierarchy being the division into PoPs (Points of Presence). Our approach, COLD (Combined Optimized Layered Design), is aimed at mirroring this design process.

In this paper we focus on the synthesis of the PoP-level, as the first step of a layered approach to generating a more detailed router-level network. 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 [5], or structural methods [6]);
- networks change less frequently at the PoP level than at the router level [7]; and
- the PoP level is the more interesting level for many activities [7], 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. 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 from the router-level optimization considered in [1]. At the PoP level, we have different constraints – we are no longer limited by technological issue such as port numbers – and the optimization objectives are different, so a considerable part of this paper is concerned with framing, and then solving, this problem (see Sections 3-7).

There are additional 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 [8, 9].
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 [1].
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 an abstract 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.

COLD satisfies these requirements. The key is choosing a synthesis process that parallels the real network design used by many network engineers, *i.e.,* heuristic optimization of economic and demographic objectives and constraints, followed by templated design to implement physical router-level structures within PoPs (see [6] for example), mimicking the highly structured and pattern-based methods recommended in basic texts on network management [2–4]. Likewise, simple heuristics can be used to create interdomain connectivity between networks if such is needed. Although this paper focusses on the PoP-level construction, our Matlab code available at [10] implements both the PoP-level algorithms described here, along with extensions to create router-level, and inter-AS level networks, and these will be described more fully in subsequent work.

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.

The result is a conceptually-simple intuitive method for generating "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 wishes primarily 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.

2. BACKGROUND AND COMPARISONS

In this section we evaluate and compare previous methods to the list of goals outlined in the Introduction. Table 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 dK-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 these graphs don't even meet simple technical constraints like connectivity. Further, these graphs do not come with any additional details such as link capacities.

Power-Law Random Graphs (PLRGs) [11] address the observed power-law node degree distribution of networks in measurement studies (at router- or AS-level). The method is simple, involves few parameters, and the general class of such methods allows tuning of the average degree and the degree-distribution, but has limited control of properties

	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 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.

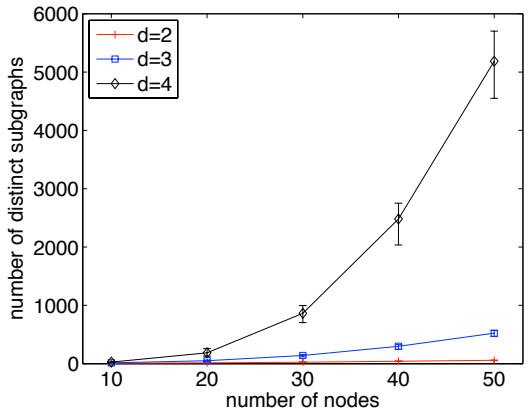


Figure 1: An example of how the number of parameters for dK-series grows rapidly both with the size of the graph and with d.

such as assortativity. Furthermore, these models have come under criticism [1,12,13] 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.* [1] 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 (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 [1]. In the foremost examples of this approach, Mahadevan *et al.* [14,15] provide 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 synthesising topologies. The 2K-distribution is used to replicate the commonly seen *assortativity* statistic, as well as the entropy statistic used in [1]. 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-distribution becomes more and more specific, increasingly restricting the graphs that match it; the set of graphs matching G in 1K-distribution is a subset of those that match the 0K-distribution, the set of graphs matching the 2K-distribution is in turn a subset of those that match the 1K-distribution, *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, such as measurement of the large number of parameters involved. Figure 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 nodes n, or possible edges in the graph. That is, the dK sequence is a longer specification than just listing the edges of the graph.

Moreover, though some of the parameters have meaning to graph theorists, they have little meaning to network engineers. Others have little meaning to anyone.

Finally, there is a more subtle problem. The dK-series can so overconstrain the problem that there is only one possible graph that can be generated. This problem is hidden by the graph isomorphism problem – it is difficult to determine if two graphs are isomorphic (*i.e.,* effectively identical). Design-based approaches avoid this problem by providing nodes with meaningful labels (*ex post facto* addition of labels doesn't solve the problem because such labels are not associated with network properties such as capacities, and if these are added as well, the resulting generation techniques lose any shred of simplicity).

The issue is illustrated in Figure 2, which shows a simple input graph, along with some Erdös-Rényi graphs with the same number of links, and 3K-series graphs generated to match it. Upon careful consideration it should be clear that the only possible 3K graph that can match the input is isomorphic to the input itself, though this might not be so obvious if the outputs were not aligned as in the figure.

This problem is far from a trivial issue. There are many examples where a given input graph completely determines

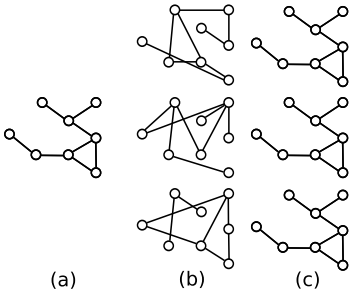


Figure 2: (a) A small example network. (b) Erdös-Rényi graphs based on that network - they all have the same number of links but in random places. One graph is not even connected and the others have very long shortest paths between some pairs of nodes. (c) graphs with the same 3K-distribution as in (a).

the output, *e.g.,* both cliques and rings. 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 subnetwork and hence only trivial differences. These are even harder to test for.

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. While it may not be exactly what one specific operator does, 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 [16].
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 optimization 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. POP-LEVEL SYNTHESIS

We focus here on synthesizing a single network, such as a network run by a single Internet Service Provider (ISP). In an individual network a single set of design rules have the most influence, because the network is under the control of a small group of designers all following a common process. Inter-AS design is impacted by decisions of multiple organizations and has aspects of game theory.

Most of the optimization steps in network design concern the PoP-level. The internal design of PoPs is almost completely determined by simple templates [2–4], since the cost of internal links is much lower than inter-PoP links. Here we focus on the PoP-level.

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 actual design problem for a large network is very complex, and costs depend on long-term business relationships with vendors. And 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, and we cannot hope to mimic all of the details of every type of heuristic used through the history of networking.

Hence, the guiding principles of the optimization are:

1. We must mirror the real-life process of designing a network, though acknowledging that it is mirrored in goals and constraints, not the exact detail.
2. The process needs to be *tunable*. As previously mentioned, real networks come in a wide variety [16], determined by different underlying cost/benefit tradeoffs. We aim to be able to tune the input parameters of our process to replicate this wide range.
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.* [17] 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 [9, 15].

There are several parts to an optimization scheme:

1. The input or *context* of the optimization problem: the PoP locations and the traffic matrix. The generation of these is described in §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 §3.2.
3. The algorithm for choosing an optimal network topology, which we describe in §3.3.

3.1 Context

It is not strictly correct to divide the area of network synthesis into *random* 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 PoP locations and traffic matrices. Thus, the generated networks are guaranteed to be different. It would certainly be possible to choose PoPs according to real-life city locations, but providing the desired variety in context would then further thought.

PoP-locations are chosen according to a 2D point process on some region. We started by testing a number of alternatives:

- different region shapes, for instance rectangles with different aspect ratios; and
- different distributions of PoP locations, for instance, we experimented with making the locations of PoPs correlated so as to make them more or less bursty.

Our experiments found, suprisingly, that these details had comparitively small affects on the resulting networks (see §7 for details). A region had to be quite long and thin before it changed the resulting networks significantly, and likewise, even highly bursty PoP locations only changed the statistics of the resulting networks slightly.

In keeping with the underlying philosophy of this work – keep the model as simple as possible – the model presented here selects n PoP locations independently, and uniformly at random on the unit square. The result is a 2D Poisson process conditional on the number of PoPs. The behavior of such a process is mathematically tractable and very well understood. However, our code is designed to be modular, so that users can use our bursty model for PoP locations, or other region shapes. In fact, it is easy to write your own module for this component, or use real PoP locations if required.

Our traffic matrix is created using a *gravity model*, proposed in [18–20], and tested [21] as a model for synthesizing traffic matrices. It suffers from identifiable flaws [20], but matches the distribution of real traffic matrices well [21]. Moreover, it is the maximum entropy model for traffic matrices under the circumstances described here [22]. The gravity model is created 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 (and 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 was small (see §7 for details).

By default we choose the simpler exponential model in most of the subsequent work, though once again it is trivial to alter this, and our tool provides this option.

It may seem counter-intuitive that the selection of context provides the randomness in our synthesis model, but that the results are not sensitive to the actual model chosen. However, note that we are not saying that the PoP locations, and the traffic don't matter. We are saying that the ensemble of networks generated is statistically similar for different context models. The individual networks generated by different instantiations of contexts are still quite different.

The statistical insensitivity of the resulting networks to the context model is interesting, but note that it only applies at the PoP-level. When we consider, for instance, the router-level network, then it is far more dependent on the traffic matrix model. A Pareto model will generate a wider spread of traffic volumes per PoP, and as a result PoPs will have a wider spread in the numbers of routers needed than in the exponential model. This insensitivity is another example of the advantage of starting synthesis at the PoP level.

3.2 Formulation of the optimization problem

Each candidate PoP-level topology is represented by an undirected graph $\mathcal{G}(N, E)$ with nodes N (the PoPs) and edges, E (the links between PoPs). The variables in our optimization are thus the locations of the links and their capacities w_i. The main constraint in the problem is that the capacities of the network are sufficient to carry the inter-PoP traffic which implicitly requires the network to be connected. We do not include redundancy, port numbers or other complex constraints at this level.

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.

3.2.1 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 overfitting the model to one particular setting, reducing its generality, and making it less applicable to, for instance, future networks.

As a result, we use a linear cost model, which is as simple and general as possible, while allowing it to be tuned to produce a wide variety of networks. 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 required on link i to carry the traffic routed across this link, and k_0, k_1 and k_2 are constants giving

0. k_0: The cost for existence of the link.
1. $k_1\ell_i$: The 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. $k_2\ell_i w_i$: The bandwidth cost, representing the cost of a given capacity over the length of the link. This term includes operating expenses, as well as initial expenditure on fiber/copper, repeaters, *etc.* [23].

Real costs have discontinuities and non-linearities (*e.g.,* a discount on the per-unit-length cost when buying longer links), but these make the model more complex, and the resulting optimization harder.

The linear cost model also has the advantage that we can rewrite the bandwidth component of cost

$$\sum_{i \in E} k_2\ell_i w_i = Ok_2 \sum_{r \in R} t_r L_r, \tag{1}$$

where R denotes the set of routes, L_r is the length of route $r \in R$, and t_r is the traffic along the route. The constant O

is the factor by which the capacity will exceed the required bandwidth, constant across all links for the sake of simplicity. This does not feature in the optimization formulation (2) since the same factor will be used regardless of which topology is chosen.

As a result, we will make the natural choice of shortest-path routing in the model, which will minimize the length of routes, and hence the bandwidth dependent component of cost. This speeds up the calculation of the required capacity, w_i, for each link, which is the dominant computational cost in the optimization process. It also means that the synthesis process provides routes as well as link locations and bandwidths. Shortest path routing is also at the core of what ISPs actually do, with tweaks to length values or to allow load balancing, *etc.*

3.2.2 Node Cost

The number of nodes n 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 [16]. Some are meshy, and others resemble 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 we could not get hub-and-spoke networks purely through optimizing against link costs (see Section 7).

Further examination of the networks in [16] also shows that they often have two classes of PoPs: *leaf* and *core*. Typically leaf PoPs had only one link[1] 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 [24]. Managing a small PoP with only a single router, and/or single link is much simpler than a multi-router, multi-link PoP.

3.2.3 Optimization Problem

The optimization problem is therefore:

$$\min_{G(N,E)} \sum_{i \in E} (k_0 + k_1 \ell_i + k_2 \ell_i w_i) + \sum_{j \in N_C} k_3, \qquad (2)$$

where G is the set of graphs on n nodes that have sufficient capacity to carry the traffic, and $N_C = \{j \in N \mid \text{degree}(j) > 1\}$ is the set of core 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 understand 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, the *spanning trees* are optimal solutions.
- k_1*-cost*: This is a 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 (1). Hence, when k_2 dominates the routes will be as short as possible, *i.e.,* the result will be a *clique*.
- 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 *hub-and-spoke* network.

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

Note that costs are all relative, and so there are really only three degrees of freedom in the above model. Hence, in the following we fix $k_1 = 1$, and only need set the other three costs.

3.3 Optimization Algorithm

The components of the objective functions 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 of the integer problem to the reals is not useful. Hence we solve it heuristically.

Guarantees that our solution is truly optimal are not necessary. This paper is not about perfect 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. So we have no need to find the very best possible network; only a set of good networks.

Hence we use a heuristic search algorithm, in this case a *Genetic Algorithm* (GA). It works by evaluating the objective function on a population of candidate topologies. The topologies with lower costs are more likely to be chosen to pass on their *genes* to subsequent generations of candidate topologies through crossover, mutation and direct selection. The process is repeated for many generations, with fitter topologies more likely to survive as the overall population improves until the topologies are well-adapted to the environment and the population reaches an almost-stable state.

The trick is to describe a topology in a form suitable for *genetic transcription*, and to create mutation and cross-over procedures that, for instance, preserve connectivity. Details of our solutions to these problems are given in §4 and Matlab code implementing the algorithm is available at [10].

However, it is worth noting that the choice of a GA over the alternative heuristics was motivated by the fact that they are:

1. *Flexibile*: GAs only require small adaptations to cope with changes to the objective function.
2. *Competitive:* 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 the result will be at least as good as these.

[1]Note that a *link* in a PoP-level network may actually correspond to multiple links between multiple routers. Hence, a degree 1 node in the PoP-level graph is not necessarily an indication of lack of redundancy.

3. *Non-exclusive:* 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 objectives in our search for a simple but realistic set (the results of which are given in §3.2.3).

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.

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 T 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.

4.1.1 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. We chose $a = 2$ and $b = 10$ 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. We

also tested other methods of crossover, but more complicated versions proved no more effective in efficiently finding low-cost topologies.

4.1.2 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.

4.1.3 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. It is used rarely. However, when the costs induce topologies with low numbers of links, this step becomes more frequent, although it is still rare to be forced to add more than one or two links to ensure connectedness.

5. PERFORMANCE OF THE GA

The first issue to resolve is the choice of settings for the GA to produce near-optimal topologies while managing its run-time. These settings include the number of generations T, the number of networks in each generation M, and the extent of mutation and crossover.

We choose the settings by running many examples with a fixed input (context), and testing a wide range of values. The goal is to choose values that produce topologies that are near optimal with minimal run-time. However, testing optimality is non-trivial, for the same reasons we are using a heuristic in the first place.

We start by comparing our results to the results of brute-force enumeration. This is infeasable for even moderately sized networks as the number of possible graphs is super-exponential in the number of nodes, but we at least ensure that for networks of up to 8 PoPs that the GA always finds the real optimal solution. This allowed us to obtain lower-bounds on settings such as the population size, and number of generations.

The other approach we use to test the GA is to compare its results to various heuristics. The heuristics were chosen so that they would work well for particular cost structures. For instance, we know that when the cost k_1 dominates, the optimal networks are minimum spanning trees, and so we use this to ensure that the GA can perform well for these parameter values.

We use a number of more complicated heuristics, primarily based on greedy approaches, described below. Each test algorithm starts with one hub node, and every other node a leaf node connected to it. Leaf nodes are converted to hub nodes one at a time, in such a way that the cost of the network reduces with each new hub (the way in which the hubs connect to each other varies). At every step the remaining leaf nodes are reconnected to the new closest hub node. 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 made a hub. New hubs are linked to the existing hubs greedily: picking the lowest cost connecting 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.
- *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 graph or 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).

We compared the GA 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 relative cost of the optimal solution found by each algorithm is plotted against k_2 in Figure 3, and we have similar plots against the other cost parameters. It is clear from the figure that different algorithms perform better in different circumstances (with different values of k_2, k_3).

Though specific greedy algorithms outperform the GA for some parameter values, the GA performs well over a range of parameters when $k_3 = 0$ (left), but is not as effective when k_3 (the hub cost) is larger. However, a significant advantage of the GA is that it can take, as part of its initial population, the output of other algorithms. When we include the output of our heuristics as inputs to the GA – the *initialized GA* results – we see that the approach outperforms all of its competitors over all parameter ranges tested. The greedy algorithms are relatively fast in many cases, thus running them before the GA is not a significant cost.

As a result of these comparisons, we determined reasonable settings for the GA, most notably we chose to fix T and M at 100 each. These values provide a reasonable tradeoff between performance and speed. The GA was tested for sensitivity to increases in M and T, and despite quadrupling the number of topologies and the number of generations, the GA showed at most 10% decrease in costs for all values of (k_0, k_1, k_2, k_3) tested. We could also choose to stop the GA once the relative rate of change of best cost was sufficiently low, but a choice of $T = 100$ proved to function similarly.

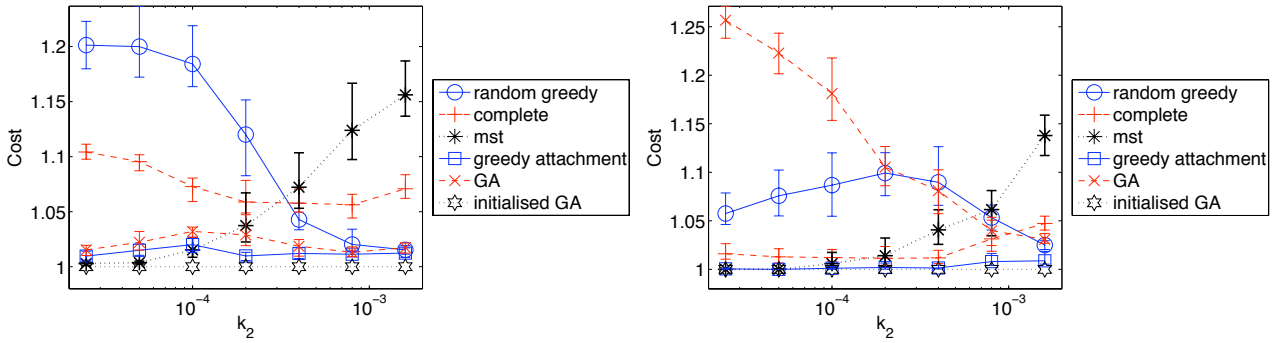


Figure 3: Cost of best solution versus k_2, normalized by initialised GA result, $n = 30$. Error bars denote 95% bootstrap confidence intervals for the mean of the results, 20 trials for each set of parameters. $k_3 = 0$ *(left)* and $k_3 = 10$ *(right)*. In both figures, $k_0 = 10, k_1 = 1$.

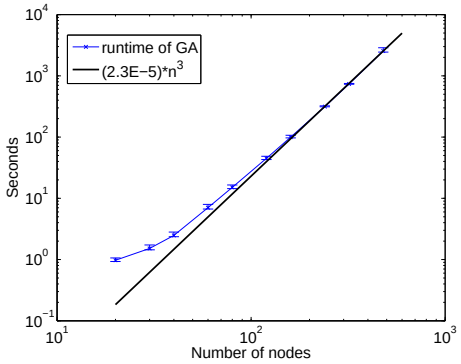


Figure 4: The run time of the genetic algorithm grows cubically in the number of nodes, with $T = M = 100$.

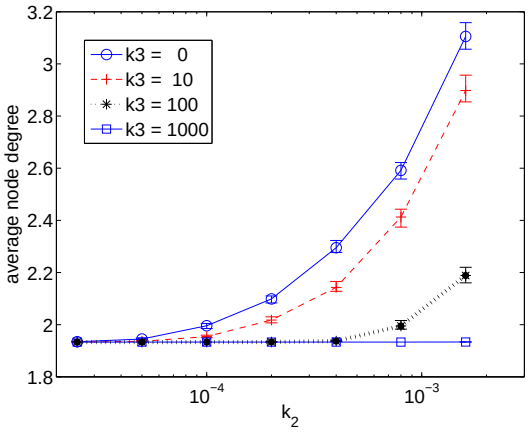


Figure 5: The average node degree versus the cost k_2, for various values of k_3, with $k_0 = 10$, $k_1 = 1$, and $n = 30$. 95% confidence intervals based on 200 simulations per data point are shown by horizontal bars.

Figure 4 shows the runtime of the GA, which grows as $O(n^3MT)$, where n is the number of PoPs, T is the number of generations and M is the number of topologies in each generation. Linearity of the GA with respect to M and T is obvious, and the n^3 term arises in evaluating the all-pairs shortest paths routing calculation for each topology.

$O(n^3)$ might seem less than ideal as an order of scaling for the run time of the GA, but given current hardware it was quite feasible to generate networks with hundreds of PoPs. This is another advantage of generating a PoP-level network: a large ISP can have thousands of routers, but it is rare to see a network with more than a 100 PoPs [7, 16]. It would be much more difficult to implement this type of optimization across all of the routers in a network.

6. TUNABILITY

One of our goals is that the output of the optimization process be tunable, as expressed in the introduction. Knight *et al.* [16] present a large set of PoP-level network maps created from data presented by network operators themselves. As a base-line, we are able to tune the output of our synthesis to produce a range of networks like those seen in this dataset. Note that we are not asserting that our networks are the same as the networks in this dataset. That is a much stronger claim that is, in point of fact, not possible to justify based on any small set of statistics (despite many claims otherwise). Rather we aim to show we can control the output of our generated networks so as to allow experimenters to test the impact of particular features of topologies on their experiments. As such, our goal is to show that for commonly studied features, we can reproduce a representative range of these features.

We illustrate here tunability with respect to average node degree, network diameter, clustering coefficient and the coefficient of variation of node degree. These encompass many of the features that have drawn research interest in the past, but we make no assertion that it is a complete set, and we have examined other features: for instance assortivity, average shortest-path lengths, and average node and link betweenness. However, the results are all of a similar nature, and the additional graphs add little additional insight.

COLD has four control parameters k_0, k_1, k_2 and k_3, but they are only unique up to a constant factor, so we can fix one (say k_1). Additionally, we found that costs k_0 and k_1 have a similar impact on the resultant networks, and so in these experiments we fix the ratio between these two by setting $k_0 = 10$ and $k_1 = 1$.

The average node degree is an important and frequently examined statistic. The higher the node degree, the more "meshy" the graph is. Intuitively, the number of links should increase as k_2 increases since it becomes more cost-efficient to have direct links between PoPs. Consequently, we expect average degree to increase with k_2. Likewise, as the cost k_3 increases, there should be fewer core PoPs, and hence fewer links, and so average degree should decrease. Figure

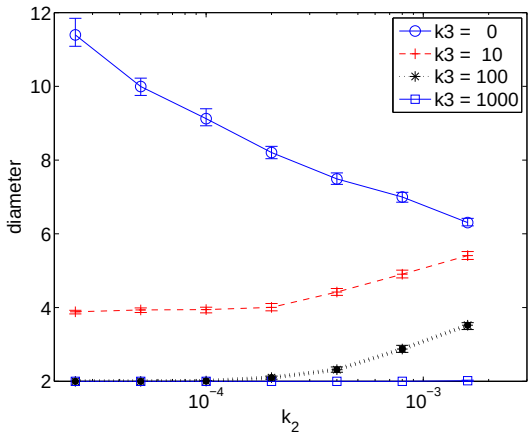


Figure 6: Network diameters for differing values of k_2 and k_3, with the same parameters as Figure 5.

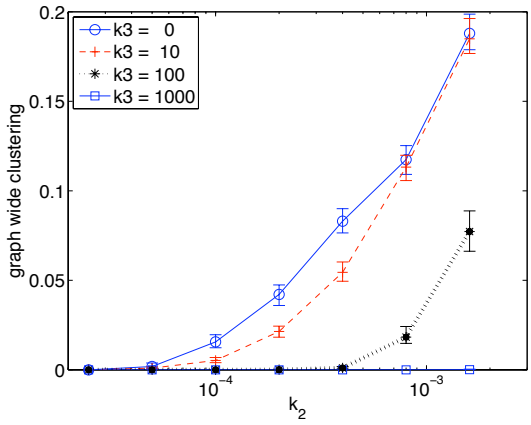


Figure 7: Global clustering coefficient for differing values of k_2 and k_3, with the same parameters as Figure 5.

5 shows exactly this behaviour. Moreover, the achievable degrees range from the minimum possible (for a tree the average degree is $2 - \frac{2}{n}$), up to the maximum (the curves continue to increase for larger values of k_2, and when k_2 dominates the optimal network is a clique).

Also of importance is the fact that the curves are smooth and monotonic, and the confidence intervals are tight around each curve, which makes it easier to generate a set of networks with well-controlled features.

The diameter of a graph is another frequently used statistic [9]. It denotes the maximum number of hops between pairs of nodes in the graph, and is important for network properties such as latency. Graphs with small diameter relative to their size and node degree demonstrate the "small-world" property, which has been of some interest in the research literature. Figure 6 shows the diameter of generated graphs with respect to k_2 and k_3. A high hub cost k_3 results in centralized networks that have low diameter. A high bandwidth cost k_2 results in meshy networks that also have low diameter. When neither cost dominates, we see higher diameters – for instance, networks with a few hubs, or networks that have too few links (especially long links) to have low diameters. Once again we can see a smooth, well-controlled progression of values, with a range that encompasses those seen in networks of comparable size in [16].

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 [16] 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 the GA to move from producing trees (with a GCC of 0) to producing fully connected graphs (with a 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 reasonable values as shown in Figure 7. However, note also that while GCC increases with k_2, the diameter can decrease with k_2, resulting in "small-world" networks.

We have examined a much larger set of network features, and different values of the ratio of k_0 and k_1, and different network sizes ($n = 50$ and $n = 80$), and uniformly we observe the same controlled variation in the statistics of the resulting networks over the observed range in real networks, or wider.

However, in our original experiments there was one statistic where the range of values observed in [16] was difficult to match, and it was this that lead us to include k_3, the hub cost, and we discuss this in more detail in the following section.

7. NODE-BASED COST

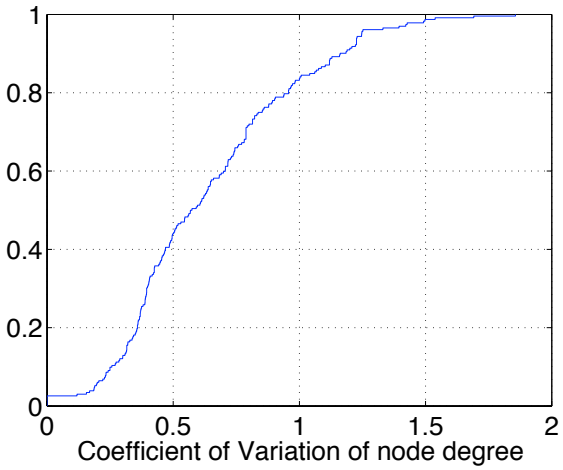
In this section we show that while it is possible to generate reasonable networks just using link costs $\sum_{i \in E}(k_0 + k_1\ell_i + k_2\ell_i w_i)$, a node-based cost $\sum_{j \in N_C} k_3$ is required to encompass all the variety we see in networks [16].

Without a node-based cost, we can vary k_2 to control the statistics mentioned above over the range seen in [16]. However, some networks in [16] have mainly PoPs of degree 1 (leaf PoPs), and few higher degree PoPs (core or hub PoPs). This "hubbiness" was measured in [16] using the coefficient of variation of node degree (CVND), which is the standard deviation of the node degree divided by the mean. Some networks in [16] have a CVND of nearly 2 (Figure 8a).

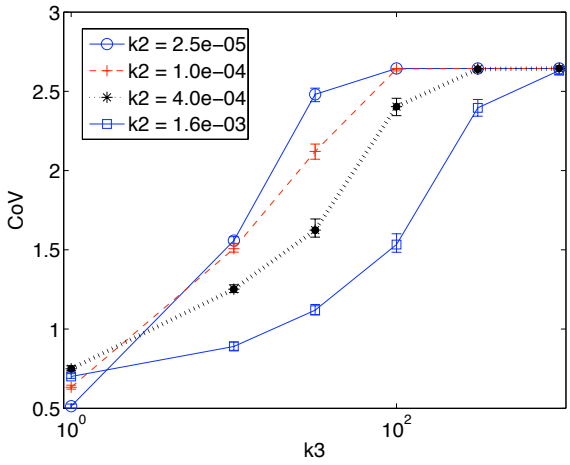
Figure 8b shows that for small k_3 (*i.e.,* the case where we don't include a hub-based cost) the synthetic networks have a CVND significantly less than one. Likewise, Figure 9 shows that the number of hub nodes is large when the hub cost is insignificant.

We need a means to induce some PoPs to be leafs. It may seem that this could be accomplished through changes in the context, *e.g.,* by making the traffic mode heavy-tailed. As noted earlier, we tried several models for PoP location generation with varying degrees of correlation, and we also considered Pareto-distributed, heavy tailed traffic models. In the later case, we trialled Pareto distributions with shape parameter $\alpha = 1.5$ and 1.1, corresponding to the infinite variance case.

All of these approaches had an effect on the generated networks. They did induce additional leaf nodes, but the effect was small. Even in the most extreme cases (bursty PoP location, and Pareto distributed traffic with $\alpha = 1.1$) the CVND was not sufficiently increased to represent the full gamut from [16].



(a) Distribution of CVND degree for networks in [16]. About 15% of the networks have a CVND over 1, a value unattainable without a node-based cost.



(b) Coefficient of variation of node degree versus k_3. Note that for small k_3, CVND is well below 1 for all cases.

Figure 8: Measured and simulated CVND, using the same parameters as Figure 5.

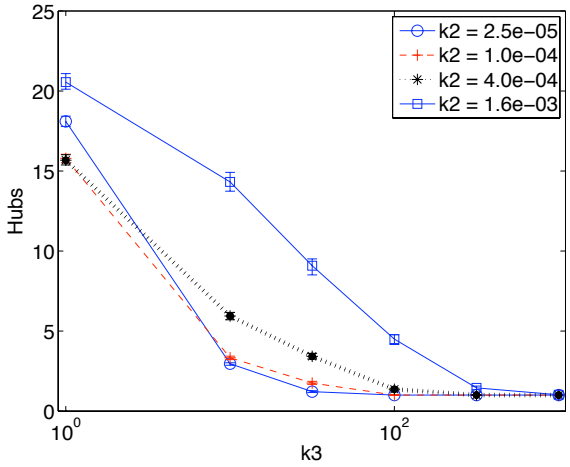


Figure 9: Number of core PoPs, for networks with 30 nodes. Again note that for small k_3 the number is always large.

The only method we found that did increase the CVND sufficiently was to include an explicit cost for non-leaf nodes. Once a hub cost is incorporated, it is possible to control both the CVND and the number of hubs, as is shown in Figures 8b and 9.

8. CONCLUSION

This paper 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.

The benefits of this type of approach include 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 [10]. This paper has focussed on the PoP-level component of this approach, we explore the other components in later work.

In the future we aim to explain in more detail how the PoP-level design rules can be exploited to perform router-level network generation and AS-level network generation. The former requires a structured approach mimicking the design rules used by network engineers, which can be expressed through graph products [25]. The latter requires elements of geography and traffic optimisation as expressed in COLD, but routing policy is also more complicated than the shortest-path routing used here.

We also plan to use statistical estimation techniques, most notably ABC (Approximate Bayesian Computation) to map real networks to parameters k_i, to assist experimenters in determining appropriate values for these parameters in specific contexts.

9. ACKNOWLEDGMENTS

This work was supported by an Australian Postgraduate Award, the Australian Research Council Centre of Excellence CE140100049 and the Australian Research Council's Discovery Projects funding scheme (# DP120102834).

10. REFERENCES

[1] 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.

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

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

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

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

[6] 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.

[7] Y. Shavitt and N. Zilberman, "A structural approach for PoP geo-location," in *IEEE INFOCOM*, 2010.

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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] P. Mahadevan, D. Krioukov, K. Fall, and A. Vahdat, "Systematic topology analysis and generation using degree correlations," in *Proceedings of the ACM SIGCOMM '06*. New York, NY, USA: ACM, 2006, pp. 135–146.

[15] 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.

[16] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," in *IEEE Journal on selected areas in communications*, vol. VOL. 29, NO. 9, October 2011.

[17] 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.

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

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

[20] 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.

[21] 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.

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

[23] K. E. Lindqvist and B. Woodcock, "Ip network architecture: Lessons learned over time." [Online]. Available: http://www.sanog.org/resources/sanog6/lindquvist-woodcock-network-architecture.pdf

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

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