COLD: Combined Optimization and Layered Design

Rhys Bowden, Matthew Roughan, Eric Parsonage, Nigel Bean, Hung Nguyen
The University of Adelaide, Australia

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 a context comprised 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, COLD, incorporating randomness and design principles to create ensembles of synthetic networks.

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: *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. Here we use the notion advanced by Li *et al.* [1] 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.

The strength of the Combined Optimized Layered Design (COLD) method presented in this paper is that it incorporates common themes from network management, but is still as simple as possible while remaining flexible and tunable.

There are several 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 [2–4].
2. A network's form and structure is driven by 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 impossible to construct, or be ridiculous [1].
3. No model is useful if its parameters cannot be estimated. Estimation is aided by a monotonic relationship between network statistics and parameters, but also assumes that the domain of the parameters encompasses all of the reasonable networks we wish our synthetic networks to resemble.
4. Model parameters must be operationally meaningful! Parameters with only abstract meaning (*e.g.,* n^{th} degree distribution) are much harder to scale correctly than meaningful parameters, *e.g.,* costs.
5. The model should generate a "network", not just a graph. Simulations need details such as link capacity, distances, or routing. These should be generated as part of the model, not as an afterthought.
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 above problems are clearly apparent when comparing networks with dramatically different goals and constraints, such as virtual networks (*e.g.,* the Facebook Social graph) and physical networks (*e.g.,* data networks). For instance, a virtual network need not be

connected, whereas a disconnected data network is broken; a virtual network has no constraint on the number of links to a node, whereas a router has a physically limited number of ports [1]; and a virtual network has little cost to adding a link, whereas physical links can cost a substantial amount, often dependent on distance. Here we concentrate only on IP-layer networks, where the costs and constraints are relatively well known.

The key to meeting these challenges is choosing a process that parallels a real network design process [5–7] used by many network engineers. We use a two stage process that starts by designing the network at the Point-of-Presence (PoP) level, using heuristic optimization. Then we use templated design to implement physical router-level structures within PoPs, mimicking the highly structured and pattern-based methods recommended in basic texts on network management [5–7].

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 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 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

The history of network research is littered with discarded network topology models. One of the earliest models was the Erdös-Rényi graph [8], where links are completely random. Waxman [9] added some spatial structure to this model with the intuition that longer links are more expensive so less likely to occur.

There is a much more extensive literature on random graphs (*e.g.,* [10–16]), 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.

However, once real data-networks were observed, 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* [10], 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 [15,17–19]. Measurements of node degree distribution suggested that this distribution was heavy-tailed, following a power-law distribution [19], irreconcilable with the structural models of the time.

Attention then focused on finding random graph models that could explain this distribution [11–16, 18, 20]. These approaches were appealing in their simplicity and the supposed universality that they predicted. Many of the models mimicked the supposed evolution of a network as it grows, though often the argument was reversed, *i.e.,* because a model generated a network with a power-law degree, it was inferred that the network evolved according to the model.

Criticisms soon appeared, ranging from comments about the problems with the data and its interpretation [21], to the fact that there are many possible networks with a power-law distribution, and only some of these satisfy other basic requirements for networks (*e.g.,* port constraints on routers) [22–25]. Various fixes have been suggested, such as the degree-sequence generators [26, 27], which add to the set of statistics use to generate a network in order to better match real networks, but these approaches introduce yet more questions about what similarity should mean.

Graphs are high dimensional objects, and comparisons of "similarity" based on a limited set of abstract features can fail simple sanity tests [1]. 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[1], of no statistical value in simulation.

While comparisons of statistics are inevitable, this is not the only criteria we need to consider. Designed networks have been shown to mimic observed statistics of real graphs, such as power-laws [1, 28, 29], 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.

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

Another underlying theme begins to emerge 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.

COLD does so by creating a random *context* for a network, then uses a (nearly) deterministic generation process that is motivated by the real goals, constraints and decisions that network engineers make in building a real network.

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, in a *tunable* way so as to be able to replicate the wide variety of networks that are observed in the *wild* [30]. Although our approach is similar to [1] in spirit, 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.

3. TOP-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 actual design problem for a large network is very complex, involving large numbers of technical constraints (*e.g.,* port limits) which depend on varying details (*e.g.,* router models). 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 [5–7], 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 §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 [30], 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.* [28] 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 [4, 26] 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 §3.1.
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 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, even if their connectivity is!

We choose n PoP locations uniformly at random on the unit square. This is the simplest approach for generating their position. 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. 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 §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 [31–33], and tested [34] as a model for synthesizing traffic matrices. It suffers from identifiable flaws [33], but matches the distribution of real traffic matrices well [34] (our main requirement). 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 §7 for details) (although it might be larger in subsequent steps, e.g., mapping PoPs down to routers §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.

3.2 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 in the network 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 which implicitly requires the network 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 §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.

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 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 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.

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

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

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 of traffic. The implication is that our cost function will be minimized by shortest-path routing.

3.2.2 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 [30]. 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 cannot get hub-and-spoke networks purely through optimizing against link costs.

Further examination of the networks in [30] 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 [35]. 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.

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 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 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, all 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* 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 *hub-and-spoke* network.

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

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 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 paper 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 Genetic Algorithm

Once we have formulated the optimization problem, we must decide on an algorithm for finding solutions. Because exact algorithms are difficult to apply to (2), 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 (breeding), mutating or surviving unchanged). The process is repeated for many generations, with fitter topologies 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 requires 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 it 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 objectives in our search for a simple but realistic set (the results of which are given in §3.2.3). We provide the details of our Genetic Algorithm in the supplied Matlab

code and the following section.

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: $k_1, \ldots, k_4$.
- The genetic algorithm parameters, including the number of chromosomes in a generation and the number of generations. More parameters appear below, in *italics*.

Outputs

- Adjacency matrix of the best topology found by the Genetic Algorithm.
- (Optionally) Adjacency matrices 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 can be fixed over all these topologies, or be different for each topology as desired.
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 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.

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, then chooses the best a of them for crossover. Example values of a and b are 3 and 7 say.

Since the each topology is a graph with n nodes, there are $\binom{n}{2}$ possible links in the new chromosome. For each of these possible links, we choose one of the 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.

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. Here *mutate_fn* is usually chosen so that m_+ and m_- tend to be much less than n.
- *Node mutation*: A node which is not a leaf node is 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 the network is left disconnected, 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.

5. PERFORMANCE OF THE GA

The first issue to resolve 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 breeding 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 to use is brute-force. While brute-force is totally infeasible for moderate 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.

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 over node a leaf connected to it. 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 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).

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. It is clear from Figure 1 that different algorithms perform better in different circumstances (with different values of k_2, k_3). 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 *initialised GA* in Figure 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 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 it is practical to generate networks of up to 800 PoPs with it (large for a PoP-level network). However, 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 10% decrease in costs for all values of (k_0, k_1, k_2, k_3) tested.

6. TUNABILITY

One of our goals is that the output of the optimization process be tunable, as expressed in points 3 and 4 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.

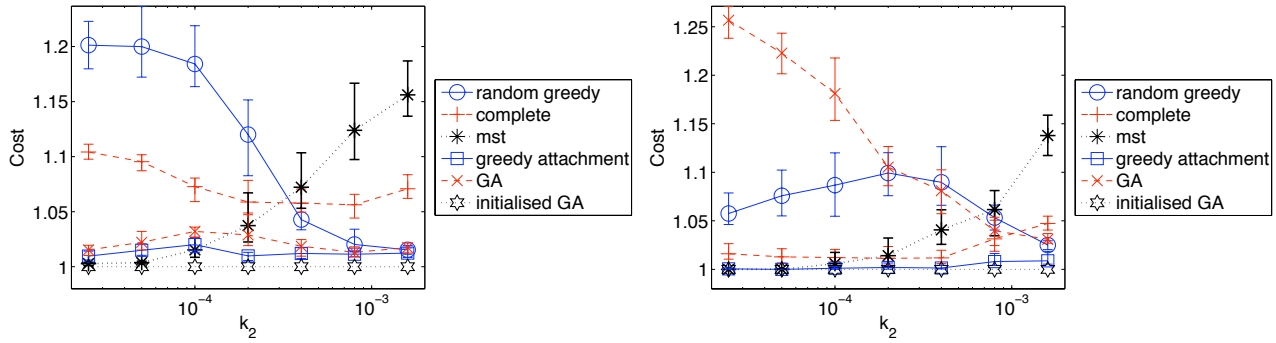

Figure 1: Cost of best solution versus k_2, normalized by GA fed result, $n = 30$. Error bars denote 95% bootstrap confidence intervals for the mean of the results, 20 trials for each set of parameters. $k_3 = 0$ *(left)* and $k_3 = 10$ *(right)*. In both figures, $k_0 = 10, k_1 = 1$.

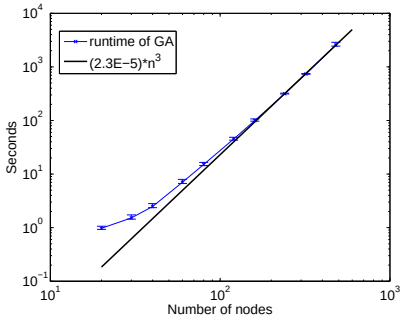

Figure 2: The run time of the genetic algorithm grows cubically in the number of nodes, with $G = T = 100$.

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 [30].

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 *exactly* equivalent optimization problem. Consequently, we fix k_0. When tuning these parameters we noticed that increasing k_0 and increasing k_1 have 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 kept the ratio between them fixed so for the following results, $k_0 = 10$ 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 the GA 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 [30], see Figures 3,4 and 5. Here we show only the graphs for $n = 30$, as they are similar for other values of n, including $n = 50$ and $n = 80$.

The average node degree is an important and 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 k_0, k_1 (and to a lesser extent k_3) costs only increase when links are added to the network, increasing them should reduce the average node degree. Increasing k_2 instead increases the average node degree, as is shown in Figure 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, 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 §3.2.3.

The majority of PoP-level networks in [30] display average node degree ranging from the minimum available, up to around 4.5 (Figure 3). The GA 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 [4]. 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 the GA can be controlled to display a wide variety of diameters. The 30 node graphs in Figure 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 graphs in [30] have diameters less than 15, and all but one have diameters less than 35.

Clustering coefficients are a way of measuring local-

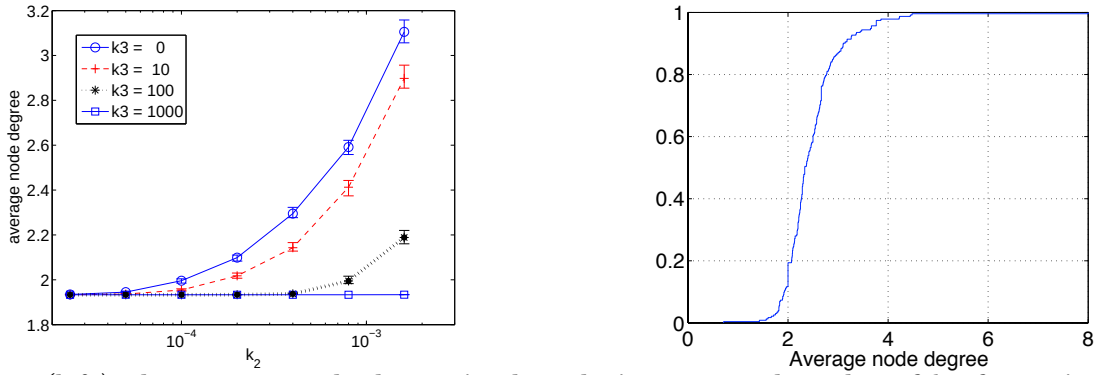

Figure 3: ***(left)*** **The average node degree in the solution versus the value of k_2, for various values of k_3, with $k_0 = 100$, $k_1 = 7$.** ***(right)*** **Empirical distribution function for average node degree of networks in [30].**

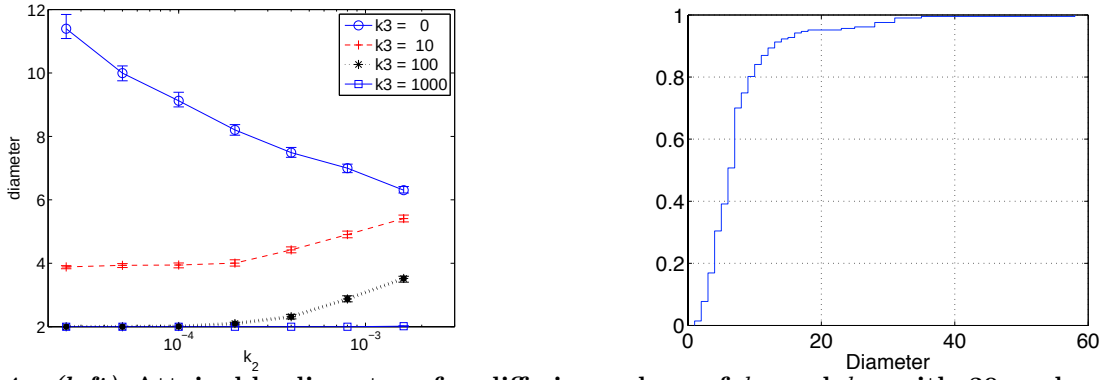

Figure 4: ***(left)*** **Attainable diameters for differing values of k_2 and k_3, with 30 node synthesized networks.** ***(right)*** **Distribution of diameter in networks from [30].**

ity, 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 [30] 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 (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 5.

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 [30].

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 [30], including diameter, average node degree, clustering, and others. However, there are some networks in [30] that have very many PoPs of degree 1 (leaf PoPs), and few higher degree PoPs (hubs). 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 [30] 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 provided to the optimization. There are two parts of the context that can be altered:

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

Usually, 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 above. 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 and infinite variance.

It is also possible to change the spatial distribution of the nodes. Normally the nodes were independently

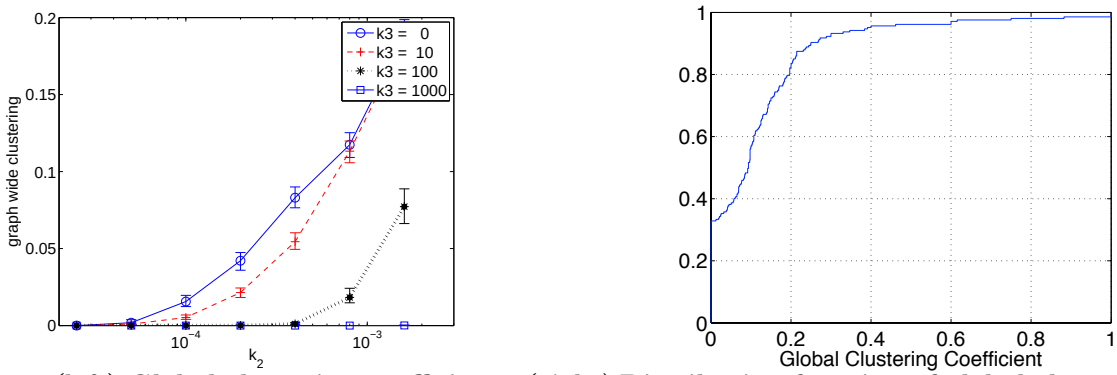

Figure 5: ***(left)*** **Global clustering coefficient.** ***(right)*** **Distribution function of global clustering coefficient for networks in [30].**

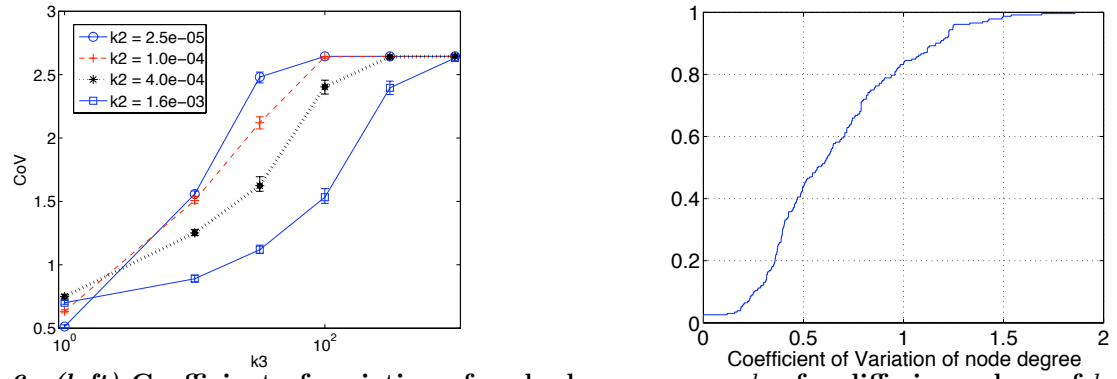

Figure 6: ***(left)*** **Coefficient of variation of node degree versus k_3, for differing values of k_2. Without a high value of k_3, it is extremely rare to produce a CVND over 1.** ***(right)*** **Distribution of CVND degree for networks in the [30]. About 15% of the networks have a CVND over 1, a value unattainable without a node-based cost.**

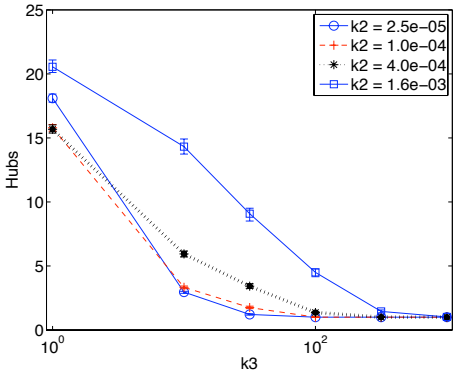

Figure 7: Number of core PoPs, for networks with 30 nodes. Plotted against k_3 for various values of k_2.

uniformly distributed over a rectangle. In order to get a few nodes with very high degrees , and many nodes with low degrees, 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 [30]. Once a hub cost is incorporated, it is possible to accurately control both the CVND and the number of hubs (Figures 6 and 7).

8. FROM POPS TO ROUTERS

So far we have been concerned with synthesizing PoP-level topologies. These are useful for some tasks, but often we need router-level topologies. In this section we describe how to build a synthetic router-level topology using a PoP-level topology as input. The method described can take input PoP-level topologies synthesized by the process described above, some other method, or even a measured PoP-level topology [36, 37].

Optimizing a network at the router-level is sometimes possible, but in practice it is rarely attempted as the problem can be much larger than at the PoP-level (*e.g.,* a moderately large network may have 40 PoPs, but 200 routers). Even when it is feasible there are additional technical and engineering constraints at the router level (*e.g.,* the complex interaction between line card options

and the number of router ports and the requirement for redundancy in parts of the network), making it is overly complex as a method for synthesizing router-level networks. Instead, COLD is inspired by the processes used by network designers.

Network operators leverage hierarchy [5–7] in network designs for two main reasons:

- *Scalability:* in many cases a hierarchical network can be made much larger than the simpler alternatives.
- *Simplicity:* hierarchy makes a network easier to visualize, and this is a key feature towards making it easier to manage. It is analogous to modularity in programming languages — ideally it allows consideration of network components in isolation.

Most frequently a network's hierarchy is based on the natural structure of its PoPs, which roughly correspond to a network 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. Using a small number of templates (repeated patterns) has many advantages: there are only a few designs for engineers to learn, 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. However, a small set of designs often suffices.

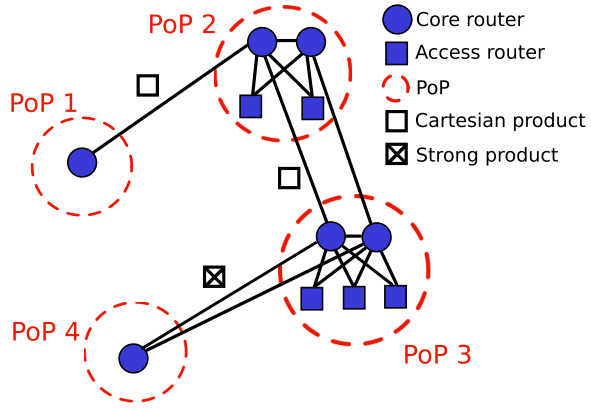

Figure 8: Each PoP shows the routers inside it and the graph product templates used to connect core routers of adjacent PoPs. PoPs with higher traffic require more access routers, and greater redundancy to nearby PoPs.

Previous work has used PoP structure in the inference of network topology [36, 37], and in network design [38]. 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.* [38], 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 [5–7] 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. Thus meeting our requirement that a network's form and structure be driven by technical constraints.

Many methods of synthesis are possible using the techniques in [38], 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 — Here to illustrate the method $M = 2$ and we choose 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.
- Generally we use K of the possible graph products: Here to illustrate the method $K = 2$, and the Cartesian product, denoted by $\square$ and the strong product, denoted by $\boxtimes$, are used to express different levels of inter PoP link redundancy.

In the language of [38], 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.

The process is shown in Figure 8, 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 generating synthetic router-level topologies [38]. 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.

9. DISCUSSION

In the introduction we presented six features that a viable network synthesis algorithm should have. COLD possesses these:

1. The networks are distinct by construction, and this is easily checked since the labels on the nodes are not arbitrary. We have also presented 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 small number of parameters and the monotonic relationship between parameters and network observables makes estimation possible. We have deliberately avoided presenting an estimation algorithm here, as to do so properly requires more space than is available.
4. The parameters are costs, which are intrinsically more meaningful than, for instance, abstract graph properties such as higher-order degree distributions.
5. COLD generates more than just a series of connected nodes. It generates PoPs with internal structure; links with capacities and distances; and routers with defined routing.
6. The PoP-level model has only four parameters, and we have shown why at least this many were needed. In particular, the hub cost is needed to create networks that could match observed instances. The simple router-level component adds only two parameters.

The last point raises one additional feature: extensibility. The two-level design makes it easy to extend COLD. The two components are not strongly coupled so we can change either one. For instance, we could use real or measured PoP-level networks for the first level, and construct a set of variable router-level networks from these.

Furthermore, the GA 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. PoPs interconnects in same cities could then be assigned a cost, and we could run the optimization with respect to this additional cost.

COLD is the only topology synthesis technique that satisfies all six of the necessary criteria defined in the introduction. Given more space we would have presented a table to demonstrate this, but there are now a very large number of alternate methods and many of them require only a cursory examination to determine their failings.

10. 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.

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 (opti-

mization and graph products); but we provide an open implementation written in Matlab [39].

11. 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] 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.

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

[4] 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.

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

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

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

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet map discovery," in *IEEE INFOCOM*, 2000, pp. 1371–1380.

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

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

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

[19] 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.

[20] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz, "Characterizing the Internet hierarchy from multiple vantage points," in *IEEE INFOCOM*, 2002, pp. 618–627.

[21] 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, www.ams.org/notices/200905/rtx090500586p.pdf.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

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

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

[33] 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.

[34] 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.

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

[36] 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.

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

[38] 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.

[39] R. Bowden, "COLD: A method for synthesising data network topologies." [Online]. Available:

http://bandicoot.maths.adelaide.edu.au/COLD/